Univerzita Karlova
Pedagogická fakulta
Katedra informačních technologií a technické výchovy

DIPLOMOVÁ PRÁCE

Programování v Pencil Code

Programming in Pencil Code

Bc. Karolína Gawlowská

Vedoucí práce: PhDr. Petra Vaňková, Ph.D.

Studijní program: Učitelství informačních a komunikačních technologií pro 2. stupeň základní školy a střední školy

2025

Odevzdáním této diplomové práce na téma **Programování v Pencil Code** potvrzuji, že jsem ji vypracovala pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Prohlašuji, že jsem při její tvorbě nepoužila nástrojů umělé inteligence jiným způsobem, než je uvedeno ve vyjádření, které je součástí textu práce. Dále potvrzuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

Praha, 13 dubna 2025

.....

Podpis

Poděkování

S velkým potěšením děkuji PhDr. Petře Vaňkové, Ph.D. za její cenné rady, ochotu a trpělivost při vedení této diplomové práce. Velmi si vážím její vstřícnosti a pochopení.

Dále děkuji kolegovi Tomáši Eiseltovi, který mi poskytl prostor pro testování úloh na základní škole. Za kompletní revizi a korekturu děkuji Nikol Zdychové.

Na závěr velké díky patří mému parťákovi za jeho neutuchající podporu a nekonečnou trpělivost.

ABSTRAKT

Tato diplomová práce se zaměřuje na rozvoj informatického myšlení žáků druhého stupně základní školy prostřednictvím tvorby grafických výstupů v programovacím prostředí Pencil Code. Informatické myšlení je v současném vzdělávacím kontextu považováno za klíčovou kompetenci, která podporuje schopnost řešit problémy, analyzovat data a vytvářet efektivní algoritmy. Využití vizuálních prvků při programování napomáhá žákům lépe porozumět abstraktním konceptům. Grafické výstupy v prostředí Pencil Code rovněž rozvíjí jejich kreativitu a poskytuje jim okamžitou zpětnou vazbu.

Teoretická část práce analyzuje různá pojetí informatického a algoritmického myšlení a porovnává dostupné nástroje vhodné pro jeho rozvoj na základní škole, se zvláštním důrazem na prostředí Pencil Code. Současně se zaměřuje na možnosti a předpoklady zařazení informatického myšlení do výuky v rámci oblasti Informatika na druhém stupni základní školy, jak je vymezeno v aktuálním Rámcovém vzdělávacím programu.

Praktická část je věnována tvorbě uceleného souboru aktivit pro žáky 7. ročníku základní školy. Soubor se skládá ze čtyř na sebe navazujících lekcí s postupně se zvyšující náročností. Tyto úlohy byly ověřeny v praxi metodou akčního výzkumu. Výsledkem šetření je vznik uceleného výukového materiálu, který obsahuje nejen samotné úlohy, ale také ukázky jejich řešení a metodické pokyny pro učitele. Součástí výstupů je rovněž webová stránka: https://gawkarol.pencilcode.net/home/index.html, na níž jsou publikovány jednotlivé úlohy.

KLÍČOVÁ SLOVA

Pencil Code, informatické myšlení, programování, 7. třída ZŠ, soubor úloh

ABSTRACT

This thesis focuses on the development of computer thinking in second grade primary school students through the creation of graphical outputs in the Pencil Code programming environment. In the current educational context, computational thinking is considered a key competence that supports the ability to solve problems, analyze data and create effective algorithms. The use of visuals in programming helps students better understand abstract concepts. Graphical output in Pencil Code also develops their creativity and provides them with immediate feedback.

The theoretical part of the thesis analyses different concepts of computational and algorithmic thinking and compares the available tools suitable for its development at primary school, with special emphasis on the Pencil Code environment. At the same time, it focuses on the possibilities and prerequisites for the inclusion of computational thinking in the teaching of Computer Science at the second level of primary school, as defined in the current Framework Educational Programme.

The practical part is devoted to the creation of a comprehensive set of activities for pupils of the 7th year of primary school. The set consists of four consecutive lessons with gradually increasing difficulty. These tasks have been tested in practice using the action research method. The result of the investigation is the creation of a comprehensive teaching material which contains not only the tasks themselves, but also examples of their solutions and methodological instructions for teachers. The outputs also include a website: https://gawkarol.pencilcode.net/home/index.html on which the individual tasks are published.

KEYWORDS

Pencil Code, computational thinking, programming, 7th grade, set of tasks

Obsah

Úvod

V dnešní digitální době je informatické myšlení nezbytnou součástí vzdělávání. Schopnost řešit problémy strukturovaným způsobem, analyzovat data a navrhovat algoritmy představuje nejen klíčovou dovednost pro oblast informačních technologií, ale také užitečný nástroj pro rozvoj logického myšlení a analytických schopností žáků. Rámcový vzdělávací program (RVP) důrazně podporuje rozvoj informatického myšlení, a proto je důležité hledat efektivní metody jeho implementace do výuky.

Jednou z možností, jak rozvíjet informatické myšlení, je využití vizuálních a grafických prvků při programování. Tvorba grafických výstupů motivuje žáky k objevování souvislostí mezi programováním a reálným světem, podporuje kreativitu a zároveň poskytuje okamžitou zpětnou vazbu na správnost algoritmických postupů. Programovací prostředí Pencil Code bylo vybráno jako klíčová platforma pro tuto studii právě proto, že kombinuje jednoduchost, intuitivní syntaxi a možnost vizuálního znázornění programových konceptů.

Cílem této diplomové práce je analyzovat možnosti rozvoje informatického myšlení žáků druhého stupně základní školy s využitím grafických výstupů v prostředí Pencil Code. Práce si klade za cíl nejen teoretickou analýzu problematiky informatického a algoritmického myšlení, ale také praktický návrh a ověření souboru vzdělávacích aktivit. Tyto aktivity jsou zaměřeny na rozvoj dovedností, jako je dekompozice problému, abstrakce, objevování vzorů a sekvencí a tvorba efektivních algoritmů.

Práce je rozdělena do několika částí. Nejprve se věnuje teoretickém východiskům informatického myšlení a jeho důležitosti v současném vzdělávacím kontextu. Následuje analýza různých nástrojů pro rozvoj algoritmického myšlení vedoucí k tvorbě grafických výstupů a jejich porovnání s prostředím Pencil Code. Dále je představen ucelený soubor aktivit, který bude pilotně ověřen v edukační praxi. Závěrem práce shrnuje dosažené výsledky a poskytuje metodická doporučení.

Spojením informatického myšlení a vizuálních prvků může tato práce přispět k efektivnější výuce programování na základních školách a podpořit žáky v hlubším porozumění informatickým konceptům.

Teoretická východiska práce

Kapitola 1

1 Vymezení cílů a výzkumného pole

1.1 Výzkumný problém

Hlavním výzkumným problémem diplomové práce je zkoumání způsobu rozvoje informatického myšlení žáků druhého stupně ZŠ prostřednictvím tvorby grafických výstupů v prostředí Pencil Code. Z hlediska definovaného problému lze vyčlenit dílčí výzkumné problémy formulované do následujících otázek:

P1 Jaká jsou teoretická východiska informatického myšlení?

P2 Jaké jsou možné nástroje a metody pro rozvoj informatického a algoritmického myšlení, jejichž výsledkem je grafický výstup, a jak se porovnávají s prostředím Pencil Code?

P3 Jak lze efektivně implementovat soubor aktivit pro rozvoj informatického myšlení do výuky na druhém stupni ZŠ?

1.2 Hlavní cíl

Hlavním cílem této diplomové práce je analyzovat informatické myšlení v kontextu tvorby grafických výstupů a prostředí Pencil Code. Na základě těchto poznatků navrhnout ucelený soubor aktivit pro žáky druhého stupně ZŠ, který bude pilotně ověřen v praxi. Tento hlavní cíl je dále členěn cíli dílčími.

1.3 Dílčí cíle a úkoly

C1 Analyzovat pojem informatického myšlení v kontextu tvorby grafických výstupů.

- **Ú1** Zmapovat pojem „informatického myšlení“ metodou komparace.
- **Ú2** Analyzovat předpoklady pro rozvoj IM na ZŠ pomocí RVP

C2 Analyzovat prostředky pro rozvoj algoritmického myšlení vedoucí k tvorbě grafických výstupů a porovnat je s prostředím Pencil Code.

- **Ú1** Identifikovat dostupné nástroje pro tvorbu grafických výstupů a rozvoj informatického myšlení.
- **Ú2** Porovnat tyto nástroje s prostředím Pencil Code z hlediska jejich vhodnosti pro výuku.

C3 Navrhnout a sestavit soubor aktivit pro rozvoj informatického myšlení na druhém stupni ZŠ s využitím grafických výstupů.

Ú1 Vytvořit ucelený soubor aktivit, který podpoří informatické myšlení a algoritmické schopnosti žáků v souladu s RVP.

Ú2 Vytvořit zadání, řešení, metodické pokyny pro učitele (včetně popsání možných komplikací během výuky a návrhu jejich řešení).

C4 Ověřit navržený soubor aktivit na vybraných skupinách žáků druhého stupně ZŠ.

Ú1 Prostřednictvím akčního výzkumu pilotně ověřit aktivity v praxi.

Ú2 Během ověřování reflektovat získané poznatky a případně upravit aktivity.

Ú3 Zpřístupnit úlohy pomocí webové stránky vytvořené v prostředí Pencil Code.

1.4 Metodika

V této diplomové práci jsou využity jak teoretické, tak empirické výzkumné metody. V teoretické části jsou aplikovány metody komparace a obsahové analýzy. Praktická část je založena na empirické metodě akčního výzkumu a na metodě návrhu a tvorby vzdělávacích aktivit.

1.4.1 Komparativní metoda

Komparativní metoda je využívána k analýze a porovnání různých přístupů k informatickému myšlení, především v kontextu tvorby grafických výstupů. Srovnávány jsou vybrané nástroje určené k rozvoji informatického a algoritmického myšlení, přičemž se sledují jejich podobnosti a rozdíly. (Jandourek, 2001) Proces srovnávání zahrnuje definování objektů analýzy, identifikaci srovnávaných znaků a následné zhodnocení výsledků. (Nešpor, 2017)

1.4.2 Obsahová analýza

Obsahová analýza je jednou z analytických metod zaměřených na systematické zkoumání dokumentů, jejímž cílem je objektivně uspořádat a odhalit skryté informace v písemných materiálech různého druhu. (Nešpor, 2017) Obsahová analýza nachází uplatnění především

při zkoumání masové komunikace, vzdělávacích materiálů či kurikulárních dokumentů, kde umožňuje identifikovat důležité vzorce a trendy. (Nešpor, 2017)

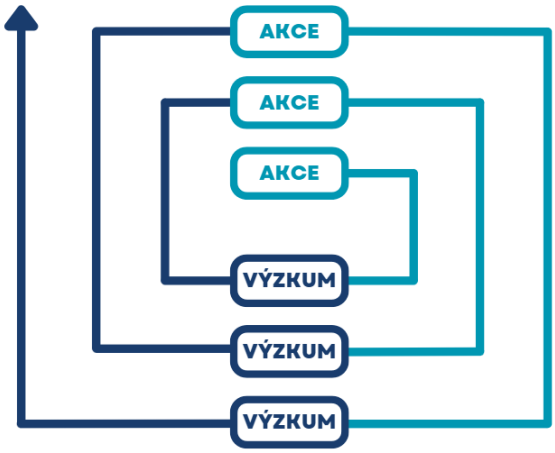
V rámci diplomové práce je obsahová analýza aplikována na studium Rámcových vzdělávacích programů, s cílem identifikovat předpoklady pro rozvoj informatického myšlení.

1.4.3 Akční výzkum

Klasická definice akčního výzkumu je ta, kterou představil John Elliott (1981, s. 1). Popsal ho následovně: *„akční výzkum je učiteli prováděná systematická reflexe profesních situací s cílem jejich dalšího rozvinutí.“*

Akční výzkum je tedy vnímán jako nástroj, který učitelům umožňuje řešit problémy školní praxe a současně realizovat inovace v této praxi. Učitel přitom vystupuje v náročné dvojí roli – jako výzkumník a zároveň jako aktivní účastník dané situace. (Janík, 2003)

Podstata a fáze akčního výzkumu dle Janíka (2003): *„Podstatu akčního výzkumu vystihují dvě slova zde obsažená: akce a výzkum. Jednoduše řečeno, akční výzkum je založen na výzkumu (reflexi) určité situace a na akci (jednání) v rámci této situace. Akce a výzkum se opakují v gradujícím cyklu (viz obrázek), přičemž akce se má neustále zlepšovat ve smyslu dosahování vyšší kvality.“*



Obrázek 1—Gradující cyklus (zdroj autorka, převzato od Janíka)

Janík (2003) uvádí, že akční výzkum slouží učitelům jako nástroj k hlubšímu pochopení procesů probíhajících v edukačním prostředí, a to prostřednictvím zkoumání a reflexe

pedagogické praxe. Jeho cílem je poskytnout učitelům prvotní porozumění složitých edukačních jevům a usnadnit tak jejich interpretaci a inovaci výuky. Na rozdíl od tradičního vědeckého výzkumu, který usiluje o hluboké porozumění a objektivní vysvětlení na základě přísných metodologických kritérií, akční výzkum se soustředí na praktické využití získaných poznatků a jejich aplikaci přímo v pedagogické realitě.

Jaké jsou tedy rozdíly mezi „klasickým výzkumem“ a akčním výzkumem? Tyto rozdíly lze shrnout v následující tabulce. (Seebauer, 2002)

KLASICKÝ VÝZKUM		AKČNÍ VÝZKUM
ZÁKLADNÍ VÝZKUM ŽÁCI JSOU "ZKOUMÁNÍ" ZÍSKÁVÁNÍ POZNATKŮ	CÍLE	ZMĚNA VLASTNÍHO VYUČOVÁNÍ UČITEL ZKOUMÁ ZÍSKÁVÁNÍ POZNATKŮ
ODVOZENY ZE VÝZKUMNÝCH ZÁMĚRŮ ODVOZENY NA ZÁKLADĚ ODBORNÉ LITERATURY	OTÁZKY	VYPLYNOU Z POTŘEB UČITELE MOHOU SE V PRŮBĚHU VÝZKUMU ZMĚNIT
STANOVEN PŘED ZAČÁTKEM VÝZKUMU	DESIGN	VYVÝJÍ SE, MŮŽE BÝT V PRŮBĚHU MĚNĚN
REPREZENTATIVA PO VÝBĚRU ZŮSTÁVÁ STEJNÝ	VZOREK	NEREPREZENTATIVA MŮŽE BÝT KDYKOLIV ZMĚNĚN
STANOVENO PŘED ZAČÁTKEM VÝZKUMU V PLÁNU VÝZKUMU	SBĚR	METODY SBĚRU DAT MOHOU BÝT KDYKOLIV ZMĚNĚNY
JAZYK VĚDY	JAZYK	JAZYK UČITELŮ
STATISTICKÉ METODY INFERENČNÍ STATISTIKA TEORIE PRAVDĚPODOBNOSTI	METODY	ANALÝZA DAT DESKRIPTIVNÍ STATISTIKA
JSOU ZOBECNĚNÉ A OBJEKTIVNÍ ČASTO JSOU K DISPOZICI TEPRVE AŽ PO NĚKOLIKA LETECH	VÝSLEDKY	JSOU SUBJEKTIVNÍ JSOU PLATNÉ „TEĎ A TADY“ JSOU BRZY K DISPOZICI

Obrázek 2—Rozdíly mezi "klasickým" a "akčním“ výzkumem (zdroj autorka, převzato od Seebauera)

Použití těchto metod umožňuje komplexní pohled na problematiku rozvoje informatického myšlení a poskytuje validní podklady pro návrh aktivit v prostředí Pencil Code.

2 Informatické myšlení

V článku Computational Thinking, Between Papert and Wing (Lodi a Martini, 2021) autoři diskutují historický vývoj a současné interpretace pojmu informatické myšlení, přičemž se zaměřují na vliv dvou klíčových postav: Seymoura Paperta a Jeannette Wing. Papertova práce, především jeho kniha Mindstorms, zdůrazňuje, že informatické myšlení je spíše o procesu učení než o konkrétních technických dovednostech. Papert věřil, že prostřednictvím programování, například v jazyce Logo, se děti mohou naučit, jak přemýšlet algoritmicky, rozkládat problémy na menší části a pracovat s tímto myšlením při tvorbě konkrétních výstupů. Wingová ve své pozdější definici tento pojem rozšířila na širší spektrum aplikací v různých oblastech, včetně vědy, inženýrství a matematiky. Lodi a Martini zkoumají, jak tyto dvě perspektivy, Papertovo soustředění na kognitivní procesy a Wingové na interdisciplinární aplikace, formují moderní pojetí informatického myšlení, které dnes spojuje jak technické, tak kreativní přístupy k řešení problémů. (Lodi a Martini 2021)

Autoři Barr a Stephenson (2011) ve své práci systematicky identifikovali základní koncepty a dovednosti spojené s informatickým myšlením. Zároveň ukázali, jak lze tyto prvky začlenit do vzdělávacích činností napříč různými obory. Přehledně shrnuli tyto principy ve formě tabulky, která nabízí praktické příklady využití informatického myšlení nejen v informatice, ale i v dalších vzdělávacích oblastech.

Podle Furbera proniká informatické myšlení do všech vědních oblastí a mění způsob, jakým jednotlivé disciplíny uvažují a pracují. Nejde již pouze o používání počítačů jako nástroje, ale o hlubší proměnu myšlenkových procesů daných oborů. Informatické myšlení Furber chápe jako schopnost rozpoznat prvky výpočetních principů ve světě kolem nás a využít nástroje a techniky informatiky k porozumění a analýze přírodních i umělých systémů a procesů. (Furber, 2012)

Podle Kempa je informatické myšlení klíčovou dovedností, kterou by si měli osvojit všichni žáci, pokud se mají efektivně uplatnit v digitálním světě a profesním životě. Shrnuje jej jako proces zahrnující dekompozici, rozpoznávání vzorů, abstrakci, zobecňování vzorů a návrh algoritmů. (Kemp, 2014)

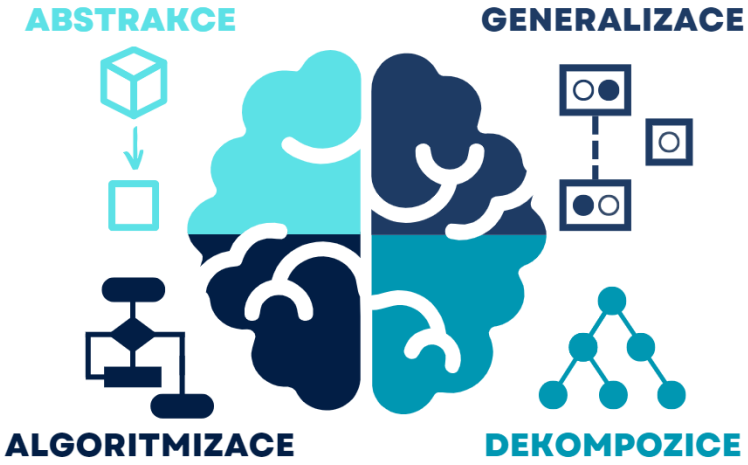
V českém prostředí je informatické myšlení chápáno jako způsob uvažování zaměřený na analýzu problémů a hledání efektivních řešení. Jde o soubor metod a přístupů, které umožňují systematicky hodnotit různé varianty řešení, rozkládat složité problémy na menší části, plánovat a organizovat činnosti či identifikovat klíčové aspekty problému. Klíčovou roli hraje schopnost vytvářet přesně popsané postupy. Tyto postupy lze opakovaně aplikovat na efektivní využívání nástrojů, jako jsou programovací jazyky nebo algoritmické myšlení. Informatické myšlení se nejprve rozvíjí na jednoduchých úlohách a postupně se aplikuje na stále složitější problémy, kde se jeho přínos pro optimalizaci řešení ukazuje nejvýrazněji. V běžném životě se uplatňuje například při organizaci rutinních činností nebo v komplexních situacích, jako je optimalizace řetězových transplantací. Tento způsob myšlení podporuje kreativitu, vytrvalost, spolupráci a schopnost efektivně komunikovat, čímž se stává klíčovou kompetencí pro moderní digitální společnost. (Imyšlení, 2018)

2.1 Informatické myšlení dle serveru BBC

Informatické myšlení (IM) je technika řešení problémů, která napodobuje proces, jímž procházejí počítačoví vývojáři při psaní počítačových programů a algoritmů. Tento proces vyžaduje, aby tito odborníci rozdělili složité problémy a scénáře na kousky, které lze plně pochopit, aby pak mohli vyvinout řešení, která jsou jasná jak pro počítače, tak pro lidi. Stejně jako vývojáři tedy i ti, kteří používají techniky informatického myšlení, rozdělí problémy na menší, jednodušší fragmenty a poté nastíní řešení, která řeší každý problém v termínech srozumitelných každému člověku. Složitý problém je takový, který na první pohled neumíme snadno vyřešit. (Brooks, 2024)

Informatické myšlení vyžaduje:

- důkladně zkoumat a analyzovat problémy, pro jejich plné porozumění.
- používat přesný a podrobný jazyk k nastínění problémů i řešení
- používání jasného uvažování v každé fázi procesu (Brooks, 2024)



Obrázek 3—Složky informatického myšlení (zdroj autorka)

Informatické myšlení spočívá v tom, jak je složitý problém uchopen a rozložen na řadu menších, lépe zvládnutelných problémů **(dekompozice)**. Na každý z těchto menších problémů se pak můžeme podívat samostatně, zvážit, jak byly podobné problémy řešeny dříve **(generalizace)**. Dále je zapotřebí soustředit se pouze na důležité informace a ignorovat nepodstatné detaily **(abstrakce)**. Poté lze navrhnout jednoduché kroky nebo pravidla pro řešení každého z menších problémů **(algoritmizace)**. (BBC, 2024)

Nakonec jsou tyto jednoduché kroky nebo pravidla použity k programování. Stručně řečeno, informatické myšlení vede lidi k tomu, aby přistupovali k jakémukoli problému systematicky a aby vypracovávali a formulovali řešení v termínech, které jsou dostatečně jednoduché na to, aby je mohl provést počítač – nebo jiný člověk.

Dekompozice

Než mohou počítače vyřešit nějaký problém, je třeba mu porozumět a způsobům jeho řešení. Dekompozice spočívá v převodu složitých problémů na jednotlivé menší tím, že složité problémy rozkládá na lépe zvládnutelné části. Menší části pak lze zkoumat a řešit nebo navrhovat samostatně, protože se s nimi jednodušeji pracuje. Pokud problém není rozložen, je mnohem těžší ho vyřešit. (BBC, 2024)

Příklad dekompozice–tvorba plakátu

S ohledem na zaměření práce na propojení informatického a algoritmického myšlení s grafickými výstupy bude v této podkapitole na příkladu tvorby plakátu ukázáno, jak lze

princip dekompozice využít k efektivnímu rozložení složitého návrhu na menší, lépe zvládnutelné kroky.

- Identifikace cílů plakátu: rozdělte úkol vytvoření plakátu na menší části. Co chcete sdělit? Kdo je cílová skupina? Jaké jsou hlavní prvky obsahu?
- Rozdělení obsahu: analyzujte obsah plakátu a rozdělte ho do samostatných částí, jako jsou titulek, text, obrázky, grafické prvky,
- Rozdělení designových úkolů: rozdělte designové úkoly na jednotlivé komponenty plakátu, například definování vizuálního stylu, výběr barev a fontů.

Obrázek 4—Příklad dekompozice u plakátu (zdroj autorka)

Generalizace

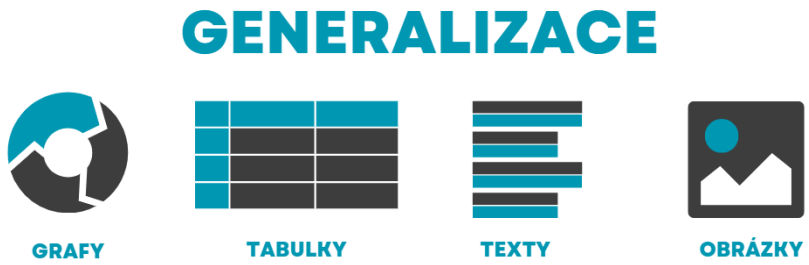
V případě rozkladu problému je často možné nalézt vzory již v menších vytvořených podproblémech. Vzory představují charakteristiky nebo podobnosti, které mají některé problémy společné. Rozpoznávání vzorů je jedním ze čtyř základních kamenů informatiky. Hledání vzorů je nesmírně důležité. Vzory nám usnadňují práci. Identifikace vzorů umožňuje zjednodušení a urychlení řešení. Pokud se v problémech vyskytují opakující se vzory, je možné aplikovat stejná řešení. Čím více vzorů se dokáže najít, tím jednodušší a rychlejší bude celkový úkol řešení problémů. (BBC, 2024)

Příklad generalizace–tvorba infografiky

V kontextu propojení informatického a algoritmického myšlení s grafickými výstupy tato podkapitola představí, jak princip generalizace umožňuje navrhovat univerzální struktury a grafické prvky infografiky.

- Vzorová struktura infografiky: generalizace základní struktury, jako jsou sekce pro úvod, hlavní informace a závěr, aby bylo dosaženo jasného a logického uspořádání.

- Všestranné rozložení: navržení univerzálních rozložení pro různé druhy dat, které budou prezentovány.
- Standardizované grafické prvky: vytvoření návrhových vzorů pro grafické prvky, jako jsou grafy, tabulky, textové pole, obrázky, ikony a diagramy, což zjednoduší a zrychlí proces tvorby.



Obrázek 5—Příklad generalizace u infografiky (zdroj autorka)

Abstrakce

Jakmile budou rozpoznány vzory v našich problémech, použije se abstrakce. Tímto krokem dojde ke shromáždění obecné charakteristiky a odfiltrování detailů, které k řešení našeho problému nejsou potřebné. Abstrakce je proces ignorování vlastností vzorů, které nejsou potřeba, abychom se mohli soustředit na ty, které jsou potřebné. Na základě toho je vytvořena reprezentace (představa) toho, co se snažíme vyřešit. To nám pomáhá utvořit si představu o problému. Pokud nedojde k abstrakci, může řešení vést k nesprávnému výsledku. (BBC, 2024)

Příklad abstrakce–tvorba infografiky

Tato podkapitola, s ohledem na propojení informatického a algoritmického myšlení s grafickými výstupy, ukáže, jak lze využít princip abstrakce k zjednodušení složitých dat a vytvoření vizuálně srozumitelných prvků, například symbolických grafů či šablon.

- Obecné téma: namísto konkrétních hodnot a dat ze zpracování, je potřeba se zaměřit na obecné tematické zaměření.
- Obecný graf: místo specifického grafu, který zobrazuje výsledky výzkumu, může být vytvořen graf znázorňující obecné vztahy nebo trendy.
- Symbolické zobrazení dat: namísto konkrétních čísel lze využít symbolické prvky, například šipky směřující nahoru k vyjádření růstu nebo dolů k poklesu.

- Univerzální šablony: zpracování abstrahuje od konkrétního vizuálního rozložení a vytváří univerzální šablony použitelné pro různorodá témata.

2.2 Informatické myšlení dle Jiřího Vaníčka

Informatické myšlení také může představovat specifický způsob uvažování, který se zaměřuje na analýzu problémů, jejich podrobný popis a hledání efektivních řešení. Tento přístup nabízí soubor nástrojů a metod, které, pokud si osvojíme, můžeme opakovaně aplikovat v různých situacích. (Vaníček, 2018)

Díky informatickému myšlení budeme schopni například:

- Systematicky hodnotit různá řešení a vybrat to nejvhodnější pro konkrétní situaci.
- Rozdělit složité problémy na menší, lépe řešitelné části.
- Plánovat a řídit různé činnosti.
- Vytvářet a podrobně dokumentovat postupy, které efektivně vedou k určenému cíli, i když je provádí jiná osoba.
- Určovat, které aspekty problému jsou klíčové pro jeho úspěšné řešení a které je možné zanedbat.
- Ukládat a strukturovat rozsáhlé a heterogenní soubory dat pro další využití.
- Používat programovací jazyky, které umožňují komunikaci s počítači, roboty a umělou inteligencí. (Vaníček, 2018)

Učení se aplikaci informatické myšlení začíná na jednoduchých úlohách a postupně přechází k složitějším problémům. S rostoucí komplexností problému se ukazuje, že nástroje a přístupy informatického myšlení jsou stále cennější. (Vaníček, 2018)

V situacích, které jsou relativně jednoduché a přehledné, se často vyplatí spolehnout na intuici. I když intuice nemusí vždy poskytnout nejefektivnější řešení, umožňuje nám reagovat rychle. Řešení problému může být v takových případech pomalejší, zahájeno je ale okamžitě. (Vaníček, 2018)

Při řešení komplexních problémů se ukazuje, že informatické myšlení je velmi užitečné. Ačkoliv se můžeme na začátku zdržet podrobným popisem a analýzou problému, tato

investice do času a zdrojů se vyplatí při následném řešení. Informatické myšlení navíc umožňuje efektivně využívat možnosti počítačů a dalších strojů řízených počítači. (Vaníček, 2018)

2.3 Informatické myšlení dle Geralda Futscheka

Informatické myšlení představuje způsob uvažování, který využívá analytické a algoritmické přístupy k řešení problémů. Lze jej chápat jako proces zaměřený na řešení problémů, v němž klíčovou roli hrají myšlenkové operace a schopnost provádět následující kroky (Futschek, 2017):

- Definovat a formulovat problém tak, aby byl řešitelný prostřednictvím počítače nebo za pomoci jiných technologických nástrojů (například hardwarových komponent).
- Logicky strukturovat, organizovat a analyzovat data.
- Reprezentovat data pomocí abstraktních metod, jako je modelování či simulace.
- Rozčlenit problém na menší části (podproblémy) a automatizovat postup řešení prostřednictvím algoritmického popisu (série kroků v logickém pořadí).
- Identifikovat, analyzovat a implementovat možná řešení tak, aby byla dosažena co nejefektivnější a nejúčinnější kombinace kroků, metod a zdrojů.
- Zobecnit získané postupy a aplikovat je na řešení dalších problémů.

2.4 Informatické myšlení dle Jeannette Wingové

Informatické myšlení je prezentováno jako univerzální přístup k řešení problémů a navrhování systémů založený na porozumění možnostem i omezením výpočetních procesů. Podle Wingové by mělo být informatické myšlení zahrnuto mezi základní dovednosti podobně jako čtení, psaní a počítání. *„Ke čtení, psaní a počítání bychom měli přidat informatické myšlení jako analytickou schopnost každého dítěte.“* (Wing, 2010)

Tento přístup je charakterizován schopností rozkládat složité problémy na zvládnutelné části a hledat jejich řešení pomocí abstrakce, simulace nebo transformace. Jsou kladeny otázky jako „Co lze vypočítat?“ a „Jaký je nejlepší způsob řešení problému?“ Wingová upozorňuje, že informatické myšlení neznamená nutit lidi myslet jako stroje. *„Počítače jsou nudné a fádní; lidé jsou chytří a nápaditi. My lidé děláme počítače vzrušujícími.“* Informatické

myšlení je představováno jako prostředek navrhování systémů a řešení problémů, které by bez výpočetních konceptů nebylo možné řešit. (Wing, 2010)

Přesah informatického myšlení je zmiňován také v jiných oborech, například v biologii, kde je využíváno k analýze dat a modelování proteinových struktur, nebo ve statistice, kde podporuje strojové učení. *„Výpočetní biologie mění způsob myšlení biologů, podobně jako výpočetní teorie her mění způsob myšlení ekonomů.“* (Wing, 2010)

Informatické myšlení tak není jen nástrojem pro technické disciplíny, ale univerzálním přístupem k problémům. Aby se informatické myšlení stalo běžnou dovedností, mělo by být součástí vzdělávání od raného věku. Wingová apeluje na pedagogy, aby učili děti přemýšlet jako informatici, a to nejen programováním, ale i pochopením výpočetních principů. *„Profesoři informatiky by měli vyučovat předmět ‚Jak myslet jako informatik‘, přístupný i těm, kdo informatiku nestudují.“* Informatické myšlení kombinuje matematický a inženýrský přístup, který umožňuje vytvářet řešení přesahující fyzický svět, a mění tak způsob, jakým lidé přistupují k výzvám moderní společnosti. (Wing, 2010)

2.5 Informatické myšlení dle kurzu na platformě Eduskop

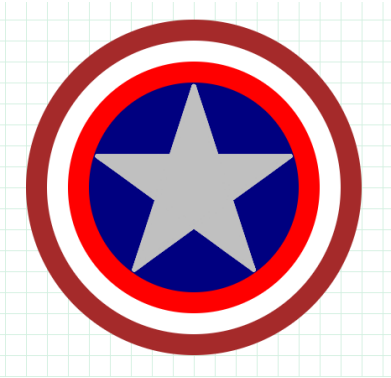
Informatické myšlení je v kurzu organizovaném platformou Eduskop prezentováno jako univerzální soubor dovedností, které nacházejí uplatnění napříč obory v 21. století. Kurz je navržen tak, aby byla účastníkům umožněna nejen znalost podstaty informatického myšlení, ale také osvojení postupů k jeho rozvoji u sebe i ostatních. Zaměřuje se na pedagogy, studenty učitelství a zájemce o moderní přístupy k výuce informatiky v České republice i zahraničí. (Eduskop, 2020)

V kurzu na platformě Eduskop je účastníkům předáváno hlubší porozumění významu informatického myšlení a jeho roli ve vzdělávání. Jsou jim představeny procesy informatického myšlení, jeho přínosy a možnosti aplikace ve výuce různých předmětů. Praktickým objevováním konceptů je podporováno pochopení toho, jak přistoupit ke změně výuky informatiky. V kurzu jsou nabízeny konkrétní příklady a strategie pro snadnější zapojení informatických přístupů do vzdělávací praxe. (Eduskop, 2020)

Dekompozice

Koncept informatického myšlení, například dekompozice, tedy princip rozkládání celku na menší části, je univerzálním přístupem, který nachází využití daleko za hranicemi informatiky či programování. Dekompozice umožňuje efektivně analyzovat komplexní problémy a nalézat jejich řešení na základě systematického postupu. Na příkladech z reálného života lze ukázat, že tyto metody nejsou pouze akademickým konceptem, ale praktickým nástrojem pro zvládání výzev, se kterými se denně setkáváme. (Eduskop, 2020)

Cílem je především demonstrovat, že informatické myšlení není omezeno na výuku informatiky. Jeho principy, jako je právě dekompozice, mohou být užitečně aplikovány i v dalších předmětech. Tím je přispěno k rozvoji kritického a analytického myšlení i napříč vzdělávacím spektrem. (Eduskop, 2020)



Obrázek 6—Příklad dekompozice problému v prostředí Pencil Code

Při rozkladu složitého obrazu na jednotlivé geometrické tvary je proces usnadněn identifikací základních prvků a analýzou jejich vzájemných vztahů, umístění a propojení. Například při analýze štítu je odhaleno, že jeho struktura je tvořena několika soustřednými kruhy: velkým vínovým kruhem, menším bílým kruhem, dalším červeným a modrým kruhem, uvnitř kterého je umístěna šedá hvězda. Tento přístup zahrnuje nejen dekompozici tvarů, ale také plánování, v jakém pořadí budou jednotlivé prvky kresleny, zejména v prostředích umožňujících kreslení ve vrstvách, jako je například prostředí Pencil Code. Je třeba zvážit, které prvky mají být vykresleny jako první, aby byly správně umístěny a překryty dalšími vrstvami. Tím je zdůrazněna nejen potřeba pochopení struktury, ale také organizace celého procesu, což je klíčovým aspektem informatického myšlení.

Vzory a sekvence

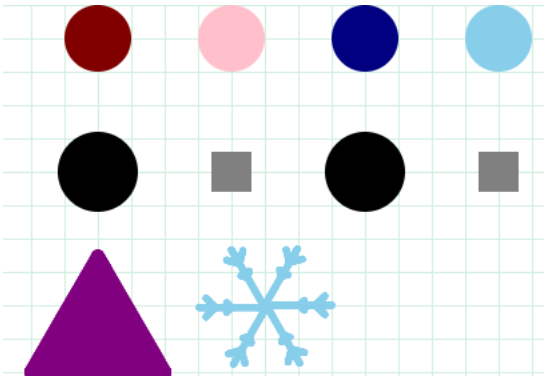
Kurz IM na platforma Eduskop vnímá rozpoznávání vzorů a sekvencí jako klíčovou schopnost, která nám umožňuje předvídat další vývoj a chápat strukturu světa kolem nás. Vzory a sekvence nacházíme nejen v matematice či informatice, ale i v každodenním životě. Rozpoznávání vzorů slouží jako základ pro pochopení logických řad a hledání analogií mezi prvky v různých skupinách. Cílem je, aby děti dokázaly identifikovat opakující se motivy a správně je doplňovat – ať už dokreslením, dolepením, nebo jinou formou. Při těchto aktivitách děti pracují s tvary, barvami, počtem prvků a učí se zachovat logiku vzoru, čímž si rozvíjejí analytické a logické myšlení. (Eduskop, 2020)

„Rozpoznávání vzorů je součástí podpory rozumových dovedností u malých dětí. Souvisí s rozvojem jak prostorových, tak matematických dovedností. Rozpoznávání vzorů zjednodušuje řešení úkolů. Problémy a úkoly se snadněji řeší, pokud odhalíme a najdeme vzory, pak totiž můžeme použít stejný způsob řešení, kdekoli se daný vzor (či vztah) objeví. Vzory zefektivňují naše myšlení a řešení problémů. Jedná se rovněž o počátek logických řad a rozpoznávání analogií na základě vztahů mezi prvky ve skupině.“
(Maněnová a Pekárková, 2020)

Pravidelnosti, vzory a sekvence jsou rozpoznávány, což umožňuje zaměřit pozornost na podstatné prvky a lépe pochopit fungování systému. Identifikací těchto vzorů jsou stanovovány předpoklady o jejich fungování, které jsou následně ověřovány a analyzovány. Díky tomuto procesu jsou odvozována obecná pravidla, která mohou být aplikována při řešení podobných problémů. (Eduskop, 2020) Odlišnosti mezi prvky jsou nacházeny prostřednictvím identifikace společných rysů. Tím je umožněno odhalení anomálií, které nejsou na první pohled patrné. Tento postup vyžaduje zaměření na detaily a rozvíjí schopnost vytrvale zkoumat rozdíly mezi prvky. (Eduskop, 2020) Při popisování struktur nebo procesů, kde se opakují vzory a sekvence, je využíváno modelování. Tento postup umožňuje vytvoření úsporného a přehledného popisu klíčových prvků. Díky abstrakci jsou zjednodušovány složité informace, aniž by byla ztracena jejich podstata. (Eduskop, 2020)

Pravidelnosti a vzory jsou využívány při tvorbě algoritmů a automatizaci postupů. Opakující se prvky jsou sloučeny, čímž je minimalizována nutnost jejich samostatného řešení. Tento

přístup přináší úsporu času, vyšší efektivitu a přesnost díky eliminaci redundantních kroků. (Eduskop, 2020)



Obrázek 7—Příklad rozpoznání vzorů v prostředí Pencil Code

Obrázek ilustruje, jak lze rozpoznávat vzory prostřednictvím střídání tvarů a barev, což je považováno za klíčovou dovednost při rozvoji informatického myšlení. V prvním řádku zobrazeného příkladu je vzor tvořen stejným tvarem – kruhem – přičemž barvy se střídají. Ve druhém řádku jsou střídány nejen barvy, ale i tvary, kdy se pravidelně objevuje kruh a čtverec, zatímco barvy a velikosti zůstávají neměnné. Ve třetím řádku je použit trojúhelník a vločka. Oba tvary jsou tvořeny vzory.

Pro rozpoznávání a pochopení těchto vzorů jsou kladeny návodné otázky jako například: „Jaké barvy a tvary se střídají?“ nebo „Jaké barvy jsou vidět a jak se opakují?“ Vzor je poté možné doplnit podle pravidla, například přidáním dalších tvarů. Takové úkoly slouží k rozvíjení schopnosti pozorování, analýzy a systematického přístupu k řešení problémů.

Abstrakce

Abstrakce je jedním ze základních principů informatického myšlení, který usnadňuje pochopení složitých problémů a systémů tím, že se soustředí pouze na podstatné rysy a potlačuje nepodstatné detaily. V kontextu platformy Eduskop se abstrakce uplatňuje například při vizuálním znázorňování objektů pomocí klíčových rysů, které jsou pro jejich identifikaci nejdůležitější. Tento přístup umožňuje žákům porozumět podstatě objektů bez zbytečného zahlcení detaily.

Jedním z běžných způsobů abstrakce je použití geometrických útvarů, které slouží k reprezentaci složitějších objektů v jednodušší formě. Taková reprezentace nejen usnadňuje

práci s těmito objekty, ale také podporuje rozvoj algoritmického myšlení. Podobně jako londýnská mapa metra neodráží přesné vzdálenosti mezi stanicemi, ale poskytuje cestujícím potřebné informace k navigaci (Berry, 2014), v informatice abstrahujeme systém tak, abychom zachovali jeho klíčovou strukturu, ale eliminovali nadbytečné detaily.

Schopnost efektivní abstrakce spočívá ve správném výběru informací, které mají být zachovány, a těch, které mohou být potlačeny, aniž by došlo ke ztrátě podstaty (CAS Computational Thinking, 2015). Tímto způsobem se žáci učí nejen principy informatického myšlení, ale také strategii efektivního řešení problémů. Abstrakce tedy spočívá v tom, že se zaměříme na klíčové aspekty problému a odstraníme zbytečné detaily. Tímto způsobem dokážeme snížit složitost úkolu a soustředit se pouze na základní prvky nezbytné pro jeho úspěšné vyřešení.

Abstrakce v Pencil Code umožňuje žákům soustředit se na podstatu řešení problému, aniž by museli řešit nepodstatné detaily. Pracují s jednoduchými reprezentacemi objektů (geometrické tvary), obecnými pohybovými instrukcemi (kroky postavy želvy), cykly a náhodností, což jim pomáhá rozvíjet informatické myšlení a pochopit principy programování na vyšší úrovni. Programovací jazyky jsou samy o sobě skvělým příkladem pro různou míru abstrakci.

Algoritmus

Na platformě Eduskop v kurzu „Jak rozvíjet informatické myšlení“ je poslední složkou tematický blok věnovaný algoritmům. Tento blok je zaměřen na praktické příklady, konkrétně na situace, kde je algoritmus chybně zadán, a učí studenty rozpoznávat, co algoritmus je a co již algoritmem není. Důraz je kladen na pochopení klíčových vlastností, které algoritmus musí mít, aby mohl být považován za správný algoritmus.

Algoritmy jsou krok za krokem instrukce, jak něco provést, nebo soubor pravidel popisujících, jak něco funguje. Recept, sada tanečních kroků nebo scénář pro animaci jsou příklady algoritmů. Učení dětí o algoritmech spočívá v tom, že je naučíme přemýšlet o krocích, které musí vykonat ony nebo počítač k dané provedení operace. (Berry, 2014)

3 Algoritmus

Algoritmus představuje jeden z klíčových pojmů informatiky, tak jeho definici nelze vyjádřit prostřednictvím jednodušších termínů. Algoritmus představuje systematický postup, který specifikuje, jak postupovat od vstupních informací ke konečnému cíli či řešení. V jeho podstatě se skládá z jednotlivých kroků, které jsou precizně definovány a vedou k dosažení požadovaného výsledku. Každý krok algoritmu má konkrétní funkci a přispívá k celkovému řešení daného problému.

Algoritmus je klíčovým pojmem v oblasti informatiky, proniká do mnoha aspektů lidské činnosti. Algoritmické postupy se běžně využívají v každodenním životě, ve vědeckém výzkumu a v průmyslové výrobě. (Chytil, 1987)

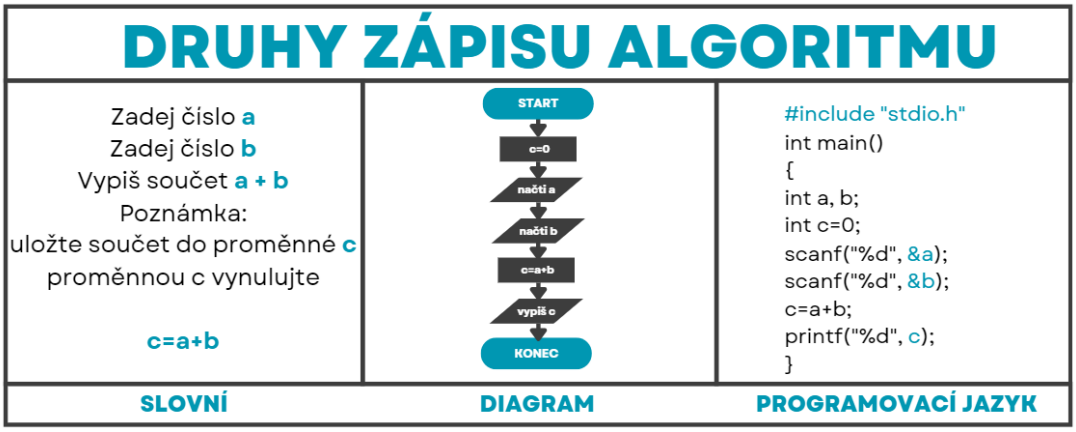
„Algoritmus je jednoznačný srozumitelný přesně specifikovaný postup (předpis) vedoucí v konečném počtu kroků k řešení určité úlohy.“ (Havelková, 2008, str. 5)

Vzhledem k jednoduššímu pojetí předchozí definice algoritmus bych ráda uvedla úplnou definici z pohledu programátora.

„Algoritmus je přesně určená konečná posloupnost instrukcí, jejichž realizace nám umožní pro přípustné hodnoty vstupních dat nalézt po konečném počtu kroků správná výstupní data.“ (Wirth 1989, cit v Havelková 2017, str. 5)

3.1 Zápis algoritmu

Způsoby zápisu algoritmu jsou různorodé. Prvním přístupem je použití přirozeného jazyka, což může připomínat návody nebo kuchařské recepty. Druhým způsobem je využití strukturovaného jazyka, který sice zůstává podobný přirozenému jazyku. Tento přístup ale podléhá určitým dohodnutým omezením v rámci používaných slovních výrazů. Třetí možností je zápis v konkrétním programovacím jazyce. Posledním způsobem pro vyjádření algoritmu je použití grafické formy, například pomocí obrázků, diagramů nebo techniky známé jako UML (Unified Modeling Language). (Štípek, 2020)



Obrázek 8—Příklady zápisu algoritmu (zdroj autorka)

3.2 Vlastnosti algoritmu

V oblasti algoritmického myšlení je klíčové si uvědomit, že ne všechny postupy nebo návody lze jednoznačně označit za algoritmy. Algoritmy jsou specifické metody řešení problémů, které musí splňovat určitá striktní kritéria. Ne každý návod či postup můžeme považovat za algoritmus. Algoritmus musí vykazovat obecnost, konečnost, determinovanost, elementárnost a resultativnost, což znamená, že jeho formulace musí být schopna systematicky a efektivně řešit různé úkoly nebo problémy. Štípek (2020) shrnuje vlastnosti algoritmu takto:

Obecnost: Algoritmus by měl být koncipován tak, aby byl aplikovatelný na celou třídu problémů, nikoliv pouze na jednu konkrétní situaci, což zajišťuje jeho univerzálnost v rámci dané kategorie úloh. Například neměl by se omezovat na daný výpočet jako 2+1, ale měl by být schopen řešit širší škálu úkolů.

Konečnost: Každý algoritmus musí mít konečný počet kroků, aby se předešlo nekonečnému cyklování a zajistilo se efektivní vyřešení problému. To znamená, že proces provedení algoritmu musí být ukončen po předem stanoveném počtu kroků.

Determinovanost: Každý krok algoritmu musí být jednoznačně definován, a po provedení každého kroku musí být zřejmé, jaký krok následuje. To eliminuje nejistotu a zajistí předvídatelný průběh algoritmu.

Elementárnost: Algoritmus by měl být složen z jednoduchých a srozumitelných kroků, které jsou elementární a snadno pochopitelné pro vykonavatele. Tato jednoduchost zvyšuje přehlednost a pochopení algoritmického postupu.

Resultativnost: Po provedení předem stanoveného počtu kroků musí algoritmus poskytnout alespoň jeden výstup. To znamená, že jeho provedení má konkrétní výsledek, který odpovídá řešení daného problému.

3.3 Algoritmické konstrukce

Algoritmy lze sestavit pomocí tří základních algoritmických konstrukcí – sekvencí příkazů, podmínek a cyklů.

Sekvence příkazů představuje jednoduchý postup, kde jsou instrukce vykonávány jeden po druhém v přesně daném pořadí. Tato konstrukce umožňuje definovat lineární postup a sledovat krok za krokem, jak se algoritmus vyvíjí. **Podmínky** umožňují vytvářet větvení v algoritmu na základě splnění určité podmínky. Jestliže je určitá podmínka pravdivá, vykoná se jedna skupina příkazů; v opačném případě se provede jiná skupina příkazů. Tímto způsobem lze algoritmus přizpůsobit různým situacím. **Cykly** umožňují opakování určitého bloku příkazů podle určité podmínky. Existují různé druhy cyklů, jako například cyklus while, který se opakuje, dokud je podmínka splněna, nebo cyklus for, který určuje přesné počty opakování. Tyto konstrukce umožňují efektivní zpracování opakujících se úloh nebo dat. (Štípek, 2020)

Tyto tři základní konstrukce jsou stavebními kameny pro vytváření složitějších algoritmů. Kombinací těchto prvků lze navrhnout algoritmy, které jsou schopny řešit rozmanité problémy a úkoly v oblasti informatiky a programování.

„V algoritmizaci a programování nejsou definice důležité (ačkoli se těm základním nevyhneme), ale schopnost analyzovat a přemýšlet. Programování a algoritmizace obecně jsou odrazovým můstkem pro řešení neobvyklých, ale i každodenních problémů. Učí nás rozložit složité operace na jednodušší, a tak lépe pochopit podstatu problému.“ (Skalka, 2007, str. 8)

Algoritmy jsou nejen omezeny na sféru informačních technologií, ale nacházejí uplatnění v každodenním životě. Často jsou spojovány s programováním a počítači, ale ve skutečnosti

jsou algoritmy jednoduše postupy nebo instrukce, které nám pomáhají řešit různé problémy. V běžném životě se setkáváme s algoritmickým myšlením ve formě návodů, postupů a kuchařských receptů. Když sestavujeme nábytek podle návodu, připravujeme jídlo podle receptu nebo následujeme manuál k novému spotřebiči, tak všemi těmito příklady aplikujeme princip algoritmu. Tyto algoritmy jsou navrženy tak, aby byly srozumitelné a jednoznačné pro běžného člověka. Každý krok je pečlivě popsán a vede nás k dosažení konkrétního cíle. I když nemusíme vždy označovat každý návod nebo postup za algoritmus, principy algoritmického myšlení jsou pevně zakotveny v našem každodenním přístupu k řešení problémů.

3.4 Algoritmizace

Algoritmizace je způsob či návrh, jak sestavit algoritmus, který splňuje všechny výše uvedené požadavky. Algoritmizace zahrnuje systematický rozbor problému, navrhování postupů a způsobu řešení, a vytváření algoritmu v odpovídajícím algoritmickém jazyce. Specifickým aspektem algoritmizace je proces sestavování a zápisu programu v konkrétním programovacím jazyce, což je známé jako programování. (Skalka, 2007, str.15)

„Proces, který provádíme při psaní algoritmu, se nazývá algoritmizace. Na začátku procesu musíme vždy určit vstupní podmínky (např. rozsah hodnot, které mohou do algoritmu vstupovat) a výstupní podmínky (vlastnosti výsledku).“ (Skalka, 2007, str.15)

„Algoritmizace je přístup k vytváření programu. Zabývá se formulací postupů řešení daného problému. Výsledkem algoritmizace je algoritmus, což je posloupnost příkazů popisující řešení daného problému.“ (Dohnal, 2008, str. 10)

Algoritmizace je tedy jednou ze čtyř základních kompetencí utvářející informatické myšlení jedince a je chápána jako schopnost algoritmického myšlení. Jedná se tedy o: *„Proces převodu problému na jednotlivé kroky, který nazýváme algoritmizace.“* (Schubert, 2011, s. 67)

3.5 Shrnutí

Algoritmus je klíčovým pojmem v informatice a představuje systematický postup, který umožňuje řešení problémů krok za krokem. K jeho správnému sestavení je důležité splnit určité vlastnosti, jako je obecnost, konečnost, determinovanost, elementárnost

a resultativnost. Algoritmy mohou být zapisovány různými způsoby, od přirozeného jazyka po programovací jazyky a grafické formy, což umožňuje jejich široké využití nejen v informatice, ale i v každodenním životě, například v návodkách nebo receptech.

Algoritmizace pak představuje proces tvorby algoritmu, kdy je důležité správně analyzovat problém, navrhnout efektivní postupy a přeložit je do konkrétního programovacího jazyka. Tento proces je součástí širšího konceptu informatického myšlení, které zahrnuje i schopnost analyzovat, strukturovat a řešit složité problémy pomocí algoritmických postupů.

3.6 Algoritmické myšlení

Dále v této části bude vymezen pojem algoritmického myšlení. Existuje celá řada definicí pojmu algoritmické myšlení. Algoritmické myšlení je způsob myšlení, který se zaměřuje na systematický a efektivní způsob řešení problémů. Toto myšlení je charakterizováno schopností rozkládat složité úkoly na jednodušší podproblémy, analyzovat a identifikovat vzory a následně vytvářet algoritmy. (Futschek, 2006)

Algoritmické myšlení je definováno jako soubor schopností, které jsou spojeny s konstruováním algoritmů a porozuměním jim. Mezi tyto schopnosti patří:

- přesné určení problému,
- analýza problému,
- nalezení základních příkazů k odpovídajícímu problému,
- sestavení správného algoritmu pomocí základních příkazů,
- přemýšlení o souvislostech,
- zvýšení efektivity algoritmu. (Futschek, 2006)

Obrázek 9—Algoritmické myšlení (zdroj autorka)

Algoritmické myšlení s výjimkou výše uvedených schopností vyžaduje i další dovednosti:

- Abstraktní myšlení: schopnost konceptualizovat a pracovat s myšlenkami a koncepty, které nemusí být přímo vizuální nebo fyzické.
- Logické myšlení: schopnost analyzovat informace a odvozovat logické závěry. Logické myšlení je klíčové pro konstrukci soudržných a platných algoritmů.
- Strukturované myšlení: schopnost organizovat myšlenky a informace systematicky, což je důležité pro vytváření jasných a efektivních algoritmů.
- Tvořivost: schopnost přistupovat k problémům s novými a inovativními pohledy, což umožňuje nalézání kreativních a efektivních řešení.
- Schopnost řešení problémů: schopnost analyzovat složité problémy a hledat systematická řešení. Tato dovednost zahrnuje schopnost identifikovat a rozkládat problémy na menší části.

Takový komplexní přístup k algoritmickému myšlení zajišťuje, že jedinec nejen chápe základní principy algoritmů, ale je také schopen efektivně vytvářet a používat algoritmy v různých kontextech. Porozumění a rozvoj těchto dovedností neslouží pouze informatice, ale také podporuje kritické myšlení a řešení problémů ve všech aspektech života. (Futschek, 2006)

3.7 Rozdíl mezi informatickým a algoritmickým myšlení

Informatické myšlení (IM) a algoritmické myšlení (AM) jsou dva úzce spjaté, ale přesto odlišné pojmy. Týkají se řešení problémů a práce s informacemi, zejména v kontextu informatiky. Informatické myšlení je širší a zahrnuje schopnost aplikovat různé techniky a přístupy k řešení problémů v oblasti informatiky. Tento přístup vyžaduje analýzu problému, jeho rozdělení na menší části a navrhování efektivních řešení, která mohou být implementována pomocí technologických nástrojů. Informatické myšlení tedy zahrnuje nejen algoritmy, ale také analýzu dat, návrh systémů a další aspekty související s počítačovými vědami.

Na druhou stranu algoritmické myšlení je specifická dovednost zaměřená na tvorbu a aplikaci algoritmů, což jsou krok za krokem postupy vedoucí k řešení konkrétního problému. Algoritmické myšlení se soustředí na strukturování problémů a hledání efektivních, jasně definovaných řešení, která mohou být automatizována. Tento typ myšlení se tedy zaměřuje na vytváření algoritmů jako součásti širšího procesu řešení problémů.

Zatímco algoritmické myšlení je součástí informatického myšlení, je to užší pojem zaměřující se výhradně na práci s algoritmy. Informatické myšlení je širší koncept, který zahrnuje jak algoritmy, tak i další aspekty – jako je návrh systémů, bezpečnost nebo analýza dat. Obě dovednosti se vzájemně doplňují a společně umožňují efektivní řešení problémů pomocí počítačových technologií.

4 Způsoby rozvoje Informatického myšlení

Předtím než jsou představeny konkrétní prostředky a nástroje pro rozvoj informatického myšlení, je klíčové porozumět postavení tohoto aspektu v Rámcovém vzdělávacím programu (RVP) pro základní vzdělávání. Tato kapitola poskytuje důkladný vhled do toho, jak je informatické myšlení začleněno do RVP ZV a jaké cíle jsou v této oblasti stanoveny.

4.1 Postavení informatického myšlení v RVP

Rámcový vzdělávací program (RVP) představuje klíčový kurikulární dokument v českém školství, definující nejvyšší úroveň vzdělávání. Původně existoval spolu s Národním programem pro rozvoj vzdělávání do roku 2020, ale od té doby prošel změnou. Na základě tohoto rozhodnutí byly vytvořeny kurikulární dokumenty na dvou úrovních – státní a školní.

Na státní úrovni se nacházejí Rámcový vzdělávací programy (RVP), které stanovují obecné rámce pro vzdělávání na jednotlivých stupních (předškolní, základní a střední vzdělávání). Tyto programy slouží jako referenční dokumenty pro vytváření školních vzdělávacích programů.

Na školní úrovni se nacházejí Školní vzdělávací programy (ŠVP), které slouží jako konkrétní nástroj pro organizaci výuky na jednotlivých školách. Tyto programy vychází z obecných principů stanovených v RVP a přizpůsobují je konkrétním podmínkám a potřebám dané školy. Oba tyto kurikulární dokumenty, ať už RVP nebo ŠVP, jsou přístupné pro širokou veřejnost, včetně pedagogické a nepedagogické veřejnosti. To umožňuje transparentnost a možnost seznámení se s cíli, obsahem a metodikou vzdělávání, což je důležité pro zapojení všech zainteresovaných stran do procesu vzdělávání. (RVP ZV, 2023, str. 5)

V kontextu informatického myšlení bylo v oblasti Informatiky zaimplementováno významné přepracování Rámcového vzdělávacího programu pro základní vzdělávání v porovnání s rokem 2017. Tato aktualizace zahrnuje nejen změnu názvu vzdělávací oblasti z Informační a komunikační technologie na Informatiku. RVP ZV explicitně uvádí algoritmizaci jako součást vzdělávací oblasti Informatika již na prvním stupni základního vzdělávání: *„Postupně si žáci rozvíjejí schopnost popsat problém, analyzovat ho a hledat jeho řešení. Ve vhodném programovacím prostředí si ověřují algoritmické postupy.“* (RVP ZV, 2023, str. 39)

Očekávané výstupy žáka prvního stupně v oblasti Informatika pro výuku Algoritmizace a programování zahrnují několik klíčových dovedností a schopností. Žák:

- *„sestavuje a testuje symbolické zápisy postupů“*
- *„popíše jednoduchý problém, navrhne a popíše jednotlivé kroky jeho řešení“*
- *„v blokově orientovaném programovacím jazyce sestaví program; rozpozná opakující se vzory, používá opakování a připravené podprogramy“*
- *„ověří správnost jím navrženého postupu či programu, najde a opraví v něm případnou chybu“* (RVP ZV, 2023, str. 40)

Očekávané výstupy žáka druhého stupně v oblasti Informatika pro výuku Algoritmizace a programování zahrnují několik klíčových dovedností a schopností. Žák:

- „*po přečtení jednotlivých kroků algoritmu nebo programu vysvětlí celý postup; určí problém, který je daným algoritmem řešen“*
- „*rozdělí problém na jednotlivě řešitelné části a navrhne a popíše kroky k jejich řešení“*
- „*vybere z více možností vhodný algoritmus pro řešený problém a svůj výběr zdůvodní; upraví daný algoritmus pro jiné problémy, navrhne různé algoritmy pro řešení problému“*
- „*v blokově orientovaném programovacím jazyce vytvoří přehledný program s ohledem na jeho možné důsledky a svou odpovědnost za ně; program vyzkouší a opraví v něm případné chyby; používá opakování, větvení programu, proměnné“*
- *„ověří správnost postupu, najde a opraví v něm případnou chybu“* (RVP ZV, 2023, str. 42)

Rámcový vzdělávací program pro gymnázia (RVP G) uvádí *„Žák se seznámí se základními principy fungování prostředků ICT a soustředí se na pochopení podstaty a průběhu informačních procesů, algoritmického přístupu k řešení úloh a významu informačních systémů ve společnosti.“ a* klade důraz hlavně na uplatňování algoritmického způsobu myšlení při řešení problémových úloh. (RVP G, 2022, str.63)

Očekávané výstupy žáka na gymnáziu v oblasti Informatika a informační a komunikační technologie pro výuku Zpracování a prezentace informací zahrnuje klíčovou dovednost a schopnost. Žák:

- *„aplikuje algoritmický přístup k řešení problémů“* (RVP G, 2022, str.65)

Rámcový vzdělávací program pro střední odborné vzdělávání (RVP SOV) oboru Informační technologie uvádí jako hlavní klíčovou kompetenci absolventů, aby: *„algoritmizovali úlohy a tvořili aplikace v některém vývojovém prostředí“.* (RVP SOV,18-20-M01, 2023, str.13)

Vzdělávání v oboru Informační technologie je zaměřeno na rozvoj schopnosti žáků využívat algoritmický způsob myšlení při řešení problémů. Žáci se učí vytvářet a formulovat postupy a řešení, která mohou být realizována jinými osobami nebo stroji, a zároveň testovat, analyzovat, vyhodnocovat, porovnávat a vylepšovat existující i navrhované algoritmy, postupy nebo informatická řešení. (RVP SOV,18-20-M01, 2023, str.50)

Výsledky vzdělávání žáka na střední odborné škole v oboru Informační technologie. Žák:

- „*rozdělí zadání nebo problém na menší části, rozhodne, které je vhodné řešit algoritmicky, své rozhodnutí zdůvodní“*
- „*navrhne algoritmy a datové struktury podle specifikace zadání a zapíše je vhodnou formou“*
- *„ve vztahu k charakteru a velikosti vstupu hodnotí algoritmy a datové struktury podle různých hledisek, porovná a vybere pro řešený problém ty nejvhodnější; vylepší algoritmus podle daného hlediska“* (RVP SOV,18-20-M01, 2023, str.52)

Z analýzy RVP vyplývá, že informatické myšlení je důležitou součástí vzdělávací oblasti Informatiky na všech úrovních vzdělávání – od základních škol přes gymnázia až po střední odborné školy. Klíčové dokumenty zdůrazňují schopnost žáků rozpoznávat, analyzovat a řešit problémy pomocí algoritmů, testovat a zdokonalovat navržené postupy a aplikovat získané dovednosti v praxi. Tato systematická podpora algoritmizace napříč vzdělávacími stupni odráží potřebu rozvíjet u žáků logické a efektivní přístupy k řešení úloh, které jsou uplatnitelné nejen v informatice, ale i v dalších oblastech jejich budoucího vzdělávání a profesního života.

5 Programovací prostředí používané k rozvoji algoritmického myšlení

Programovací prostředí pro děti hrají klíčovou roli při rozvoji informatického myšlení a algoritmických dovedností. V této kapitole budou analyzovány vybrané platformy z hlediska jejich dostupnosti, přístupnosti pro začátečníky, podpory rozvoje informatického myšlení a aktuálnosti v praxi. Na základě analýzy potřeb žáků a typů nástrojů pro výuku algoritmizace byl výběr programovacích prostředí zúžen na blokové a textové programovací jazyky s grafickým výstupem, zejména zaměřenými na kreslení pomocí programování. Analýza se zaměří na prostředí Scratch, Code with Anna and Elsa, Trinket, Turtle Academy, Khan Academy, Pencil Code a Python (želví grafika). Do tohoto výběru nebyly zahrnuty robotické hračky ani jiný programovatelný hardware. I když robotické hračky a roboti mohou být cenným doplňkem vizuálních programovacích jazyků, v této práci byla pozornost soustředěna na výuková prostředí, která využívají blokové nebo textové kódování s vizuálním výstupem.

5.1 Scratch

Scratch je vizuální blokové programovací prostředí vyvinuté MIT. Toto prostředí umožňuje dětem vytvářet interaktivní příběhy, hry a animace. Díky svému grafickému rozhraní je vhodné i pro úplné začátečníky. Mezi hlavní výhody patří široká komunita uživatelů, množství dostupných výukových materiálů a podpora kreativity. Nevýhodou je omezená možnost přechodu na pokročilejší úroveň programování, protože se nejedná o plnohodnotný textový programovací jazyk. Odkaz na prostředí: https://scratch.mit.edu/

5.2 Code with Anna and Elsa

Toto prostředí vzniklo v rámci iniciativy Code.org a slouží k výuce základů programování pomocí vizuálních bloků a populárních postav z pohádky Ledové království. Díky interaktivnímu formátu je atraktivní zejména pro nejmladší uživatele. Jeho hlavní nevýhodou je omezená hloubka výuky, což z něj činí spíše jednorázový nástroj než dlouhodobou vzdělávací platformu. Odkaz na prostředí: https://studio.code.org/s/frozen/lessons/1/levels/1

5.3 Trinket

Trinket je online platforma umožňující psaní a spouštění kódu přímo v prohlížeči. Podporuje více jazyků, včetně Pythonu, a je ideální pro experimentování s kódem. Hlavní výhodou je snadná dostupnost bez nutnosti instalace, nevýhodou pak menší zaměření na vizuální tvorbu a nižší atraktivita pro nejmladší děti. Odkaz na prostředí: https://trinket.io/

5.4 Turtle Academy

Turtle Academy se zaměřuje na výuku programování pomocí želví grafiky v jazyce Logo. Poskytuje intuitivní prostředí vhodné pro základní školu a podporuje rozvoj algoritmického myšlení. Nevýhodou je omezená využitelnost pro pokročilejší programování a zastaralost technologie oproti modernějším platformám. Odkaz na prostředí: https://turtleacademy.com/

5.5 Khan Academy-kreslení pomocí programování

Khan Academy nabízí sekci zaměřenou na programování pomocí JavaScriptu, kde se studenti učí kreslit grafické objekty kódováním. Tento přístup propojuje vizuální výstup s textovým programováním a umožňuje postupný přechod k pokročilejším technikám. Výhodou je dostupnost výukových videí a postupného vedení, nevýhodou pak vyšší náročnost pro úplné začátečníky a nutnost základní znalosti angličtiny. Odkaz na prostředí: https://cs.khanacademy.org/computing/computer-programming/programming

5.6 Python-želví grafika

Python nabízí vestavěný modul „`turtle`“, který umožňuje kreslení pomocí jednoduchých příkazů. Tato metoda usnadňuje přechod k plnohodnotnému textovému programování a je vhodná i pro starší žáky. Výhodou je spojení s reálným programovacím jazykem, nevýhodou pak vyšší náročnost syntaxe pro začátečníky a potřeba větší míry abstrakce.

5.7 Pencil Code

Pencil Code je online prostředí zaměřené na výuku textového programování pomocí JavaScriptu, HTML a CSS. Je vhodné pro přechod z blokového programování na textové, jelikož umožňuje kombinaci obou přístupů. Hlavní výhodou je možnost postupného přechodu ke složitějším konceptům, nevýhodou může být náročnější orientace pro úplné začátečníky. Odkaz na prostředí: https://pencilcode.net/

5.8 Srovnávací tabulka

Tabulka 1—Srovnání prostředí pro rozvoj IM z hlediska grafického výstupu

Prostředí	Typ programování	Dostupnost	Atraktivita pro děti	Podpora IM	Náročnost	Cílová věková skupina
Scratch	Blokové	Zdarma, online/offline	Velmi vysoká	Vysoká	Nízká	Mladší děti
Code with Anna and Elsa	Blokové	Zdarma, online	Vysoká	Nízká	Velmi nízká	Mladší děti
Khan Academy	Textové (JavaScript)	Zdarma, online	Střední	Vysoká	Střední	Starší žáci
Trinket	Textové (Python, HTML, JS)	Zdarma, online	Střední	Vysoká	Střední až vysoká	Starší žáci
Turtle Academy	Textové (Logo)	Zdarma, online	Střední	Střední	Nízká	Mladší děti
Python (turtle)	Textové (Python)	Zdarma, offline	Nízká až střední	Vysoká	Vysoká	Starší žáci
Pencil Code	Blokové, textové (JavaScript, HTML)	Zdarma, online	Střední	Vysoká	Střední až vysoká	Starší žáci

5.9 Shrnutí

Analýza programovacích prostředí ukázala, že Scratch zůstává nejrozšířenější a nejatraktivnější platformou pro začátečníky díky své intuitivní blokové struktuře a široké komunitní podpoře. Pro přechod na textové programování je však vhodnější Python s želví grafikou nebo Khan Academy, které poskytují hlubší porozumění kódu. Mezi těmito nástroji

však vyniká Pencil Code, který kombinuje to nejlepší z obou světů – nabízí jak blokové programování pro začátečníky, tak plynulý přechod k textovému kódu. Navíc umožňuje tvorbu webových aplikací v JavaScriptu, což ho činí všestranným nástrojem nejen pro výuku základů algoritmizace, ale i pro reálné projekty. Právě díky této univerzálnosti byl Pencil Code vybrán jako hlavní prostředí pro praktickou část této práce, neboť umožňuje studentům postupně rozvíjet své dovednosti od vizuálního skládání kódu až po pokročilé programování v JavaScriptu.

6 Pencil Code

Pencil Code je online programovací nástroj, který je určen zejména pro začínající programátory, děti nebo dospělé, kteří se chtějí naučit základy programování. Nástroj kombinuje jednoduchost a interaktivitu, což umožňuje uživatelům rychle začít psát a spouštět svůj vlastní kód.

Pencil Code, postavený na základech programu Scratch od MIT, představuje interaktivní webovou platformu pro kreslení, přehrávání hudby a tvorbu her. Tento nástroj rovněž umožňuje experimentování s matematickými funkcemi, geometrií, tvorbou grafů, webovými stránkami, simulacemi a algoritmy. Programy jsou otevřené a dostupné pro prohlížení a kopírování, což umožňuje sdílení tvůrčích projektů. Pro pedagogy, kteří chtějí přiblížit programování studentům prostřednictvím interaktivního a zajímavého přístupu, je Pencil Code považován za atraktivní nástroj. (Brian Ramirez, 2024)

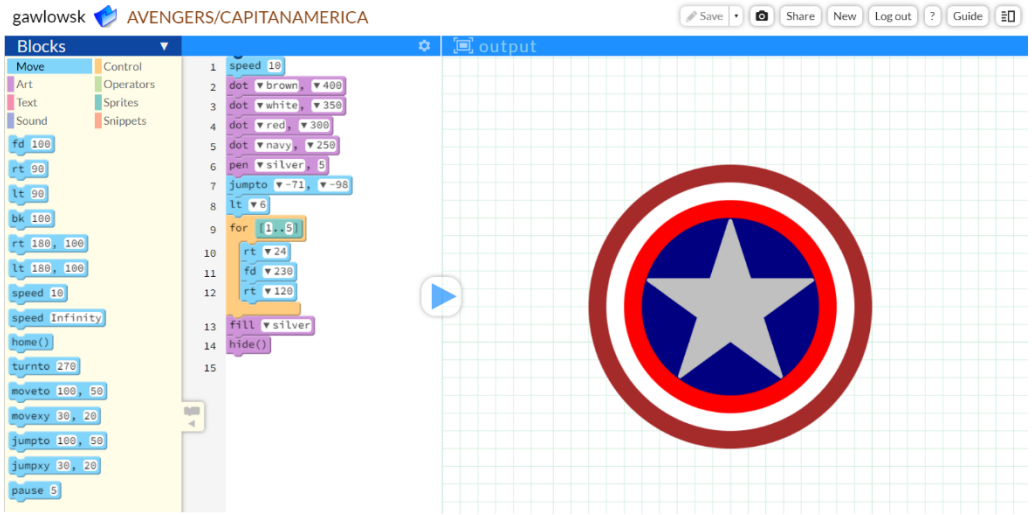
6.1 Hlavní rysy prostředí

Pencil Code nabízí flexibilní programovací prostředí, které spojuje blokové a textové programování. Blokové programování usnadňuje začátečníkům porozumění struktuře kódu, zatímco možnost přepnutí na klasický textový zápis umožňuje pokročilým uživatelům rozvíjet své dovednosti v psaní kódu. Interaktivita je jedním z klíčových rysů tohoto prostředí. Uživatelé získají okamžitou zpětnou vazbu. Po vložení a spuštění kódu ve vlastním scénáři mohou sledovat, jak jejich kód ovlivňuje obrazec želvy. Kromě toho nabízí Pencil Code bohaté grafické možnosti, které umožňují uživatelům vytvářet vizuální efekty a animace, což činí programování nejen efektivním, ale i atraktivním a zábavným. Díky formátu webové aplikace není zapotřebí instalovat speciální softwarové vybavení ani samotný Pencil Code. K funkcionalitě postačí webový prohlížeč a připojení k internetu.

6.2 Popis prostředí

Prostředí je přehledné, uspořádané a uživatel se v něm snadno a rychle zorientuje. I přestože se jedná spíše o lineární prostředí, tak i přesto nabízí nepřeberné množství možností, kterých může kreativní uživatel využít ke své tvorbě. Mohou tak vzniknout i rozsáhlé projekty. Prostředí je rozděleno do tří hlavních částí. V levé části se nachází příkazy uspořádané do různých kategorií v režimu **`blocks`**. Střední část slouží k tvorbě scénáře, kam lze

přetahovat bloky s příkazy podle principu **drag and drop**, což připomíná skládání puzzle. Je možné přepnout se do režimu **text code**, kde se píší příkazy v souladu se správnou syntaxí. Kategorie s příkazy lze snadno přepínat. V pravé části je výstup, kde se nachází obrazec želvy, která vykonává scénář neboli algoritmus. Pro určení pozice na výstupu je využívána kartézská soustava souřadnic, s počátkem v geometrickém středu výstupu. Postava želvy začíná vždy uprostřed výstupu na souřadnicích [0;0]. Pro snadnější orientaci je na pozadí výstupu zobrazena čtverečková síť. Aktuální souřadnice kurzoru myši jsou viditelné v pravém dolním rohu. (Gawlowská, 2022)



Obrázek 10—Screenshot prostředí

V Pencil Code je každý program prezentován jako webová stránka. V horní liště prostředí se nachází několik informací, které umožňují sledovat způsob, jakým je stránka publikována přes URL. V pravém horním rohu jsou umístěna tlačítka pro uložení a sdílení programu, stejně jako tlačítka pro správu webové stránky a získání nápovědy. Název programu je zobrazen vlevo nahoře a lze ho přejmenovat dvojitým kliknutím na název.

Obrázek 11—Horní lišta

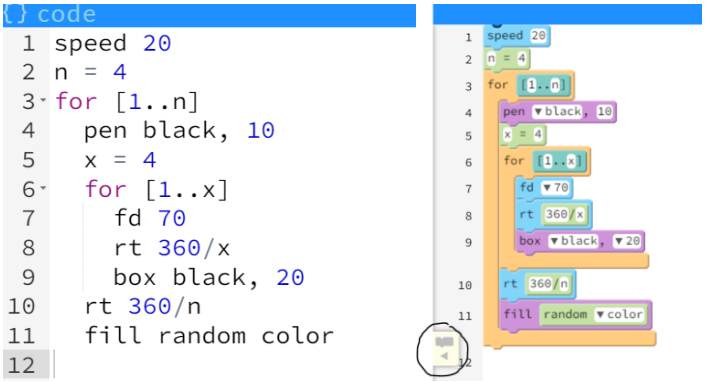
Po najetí na ikonu "**output**" v modrém pruhu vpravo nahoře je zobrazen odkaz "**open window**". Toto uživateli umožňuje otevření nové kary, na které je vyobrazen pouze výstup programu. Nezobrazuje se celé programovací prostředí. Tento odkaz je dostupný

pouze po přihlášení a uložení programu. Je dobré vyzkoušet si spuštění svých programů na celou obrazovku a odtud pomocí klávesy Ctrl+U "zobrazit zdroj" na vlastní webové stránce.

Obrázek 12—Kliknutím na tlačítko "output" se program spustí v celoobrazovkovém režimu, aniž by se zobrazil zdrojový kód.

V kódu Pencil Code obsahují výstupní adresy URL slovo "/home/". Změnou adresy "/home/" na "/edit/" se vytvoří adresa URL, která se zobrazí v Pencil Code jako editační uživatelské rozhraní.

Pencil Code umožňuje programátorovi využívat blokový režim a pomocí přetahování bloků navrhovat program. Tyto bloky jsou přímou reprezentací základního textového jazyka, jako je JavaScript nebo HTML. I když jsou bloky vizuálně odlišné od textového kódu, slouží pouze jako grafické zobrazení a úprava instrukcí v programovacím jazyce.



Obrázek 13—Screenshot textového režimu a blokového režimu

Blokový a textový režim je naprosto rovnocenný, pokud jde o výkon a vyjadřovací schopnosti. Výchozím prostředím je blokové programování, ve kterém se odehrává většina práce, avšak žáci mají kdykoliv možnost přepnout pomocí tlačítka do textového režimu a nahlédnout, jak odpovídající kód vypadá v jazyce JavaScript nebo HTML.

Pencil Code nabízí uživatelům možnost vytvářet scénáře pomocí příkazů z různých kategorií. Každá kategorie je odlišena barevně podle specifických činností (Gawlowská, 2022):

Move (pohyb, modrá): Obsahuje příkazy pro pohyb, jako je posun dopředu a dozadu, otáčení o úhel, rychlost a přesun na konkrétní pozici v souřadnicové soustavě (osa X a Y).

Art (vzhled, fialová): Zahrnuje příkazy pro definování vzhledu, vytvoření tečky a čtverečku, nastavení velikosti a barvy, a úpravy obrysu a pera postavy želvy.

Text (text, růžová): Obsahuje příkazy spojené s textem.

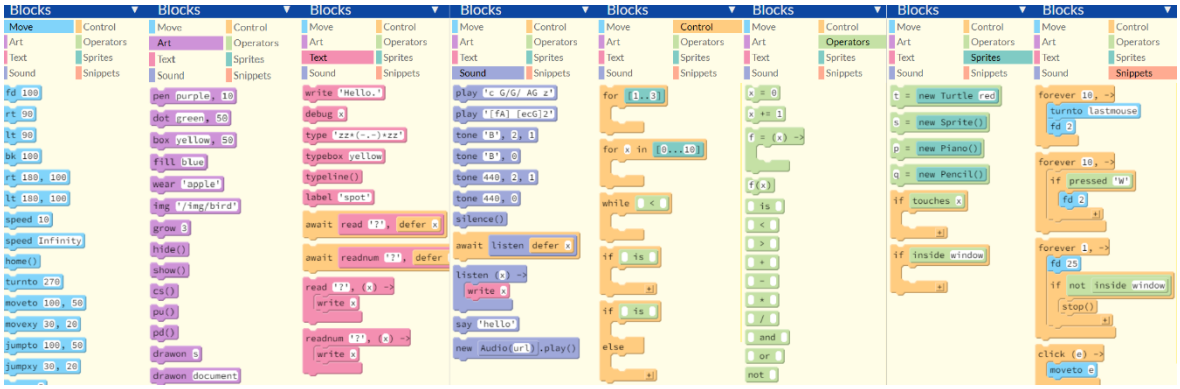
Sound (zvuk, modro-fialová): Nabízí příkazy pro manipulaci se zvukem.

Control (ovládání, oranžová): Zahrnuje bloky pro vytváření cyklů a podmínek, a také příkazy pro zachycení událostí, jako je stisknutí klávesy nebo tlačítka myši.

Operators (operátory, zelená): Obsahuje matematické a logické operátory, proměnné a bloky pro tvorbu funkcí.

Sprites (tmavě zelená): Poskytuje příkazy pro vytváření nových objektů.

Snippets (tomatová): Tato kategorie zahrnuje příklady a hotové části kódu pro inspiraci.



Obrázek 14—Všechny kategorie

6.3 Výhody prostředí

Pencil Code je online a zcela zdarma, což umožňuje žákům ihned začít s tvorbou bez potřeby stahování nebo instalace. Pro registraci stačí navštívit webovou stránku http://pencilcode.net/, kde je proces registrace velmi jednoduchý a intuitivní. Pro vytvoření účtu není nutná ani e-mailová adresa. Postačuje zadat jméno či přezdívku společně s heslem. Po dokončení registrace si žák může ukládat své projekty, což mu umožňuje pracovat na úlohách i mimo školní prostředí. Jako přínos lze ocenit animovaný pohyb obrazce želvy,

díky němuž lze sledovat detailně její pohyby a rotace. Žáci mají také možnost individuálně nastavovat rychlost pohybu obrazce želvy a poloměr její rotace.

Díky jasně barevně odděleným kategoriím se žáci snadno naučí rozpoznávat a rozlišovat jednotlivé příkazy. Kód je tvořen bloky, které se vzájemně skládají jako puzzle. Tento systém skládání bloků přináší výhodu v tom, že žák nemůže udělat chybu v syntaxi, což usnadňuje proces učení a programování (například klasicky chybějící středník na konci řádku). Pro spuštění algoritmu není potřeba provádět složité stahování nebo spouštět exe soubory, což zjednodušuje proces programování a testování kódu. (Gawlowská, 2022)

Experimentální část

Kapitola 2

7 Rozbor úloh

Celkově úlohy podporují informatické myšlení jako celek. Žáci nejprve formulují a analyzují problém, navrhují a realizují řešení, které je zapsáno jako algoritmus, a jehož výstup lze vizuálně ověřit. Důraz je kladen na klíčové operace jako abstrakce, dekompozice, algoritmizace a strukturované myšlení, které žáci uplatňují při řešení úloh a zápisu algoritmů v prostředí Pencil Code. V tomto prostředí se zároveň učí základům programování, včetně ladění kódu, přechodu mezi blokovým a textovým režimem a práce s proměnnými či cykly. Navržené úlohy tímto způsobem rozvíjejí nejen znalosti programování, ale i logické uvažování, schopnost řešit problémy a kreativní přístup.

Navržené úlohy vycházejí z požadavků Rámcového vzdělávacího programu pro oblast „Algoritmizace a programování“. Navržené úlohy mají přesah i do oblasti digitální gramotnosti a podporují mezipředmětové propojení, zejména s matematikou (např. v tématech geometrie, symetrie nebo práce se souřadnicemi). Inspirovány jsou mimo jiné i modelovými úlohami z portálu Umíme informatiku (např. ze sekce Želví grafika), které kombinují algoritmické myšlení s grafickým výstupem jako zpětnou vazbou.

Z pedagogického hlediska jsou úlohy navrženy tak, aby byly diferencované, umožňovaly postupné osvojování nových konceptů, a zároveň byly dostatečně flexibilní pro různé typy žáků i vyučovacích stylů. Obsahují zadání, vzorová řešení a metodická doporučení. Každá lekce má jasně definované výstupy. Navržené úlohy jsou koncipovány tak, aby žáci mohli dosáhnout alespoň dílčích cílů, zatímco pokročilejší mají možnost pracovat na rozšířených variantách. Tím se posiluje samostatnost a motivace. Žáci si uvědomují vlastní pokrok a rostoucí schopnost zvládat složitější úkoly. Struktura úloh vychází z principu postupného budování dovedností, kdy nové znalosti vždy navazují na předchozí zkušenosti.

7.1 Přehled úloh

Tabulka 2—Přehled úloh

Název úlohy	Skupina	Lekce	Formy/metody výuky	Náročnost
Hřib	1, 2, 3 a 4	Lekce 1	Samostatná práce	1
M	1, 2, 3 a 4	Lekce 1	Samostatná práce	2
Čtverec	1 a 2	Lekce 1	Samostatná práce	1
Slunce	3 a 4	Lekce 1	Samostatná práce	2
Schody	1, 2, 3 a 4	Lekce 2	Samostatná práce	2
Puzzle	1 a 2	Lekce 2	Samostatná práce	3
Kříž	3 a 4	Lekce 2	Samostatná práce	3
Had	1, 2, 3 a 4	Lekce 2	Samostatná práce	2
Okno	1, 2, 3 a 4	Lekce 3	Samostatná práce	4
Květina	1, 2, 3 a 4	Lekce 3	Samostatná práce	3
Kolečko z teček	1, 2, 3 a 4	Lekce 3	Samostatná práce	4
Síť šestiúhelníků	1 a 2	Lekce 4	Samostatná práce	5
Hvězda	3 a 4	Lekce 4	Samostatná práce	5
Projekt	1, 2, 3 a 4	Závěrečný projekt	Samostatná práce/ práce ve dvojicích	99

Stupně náročnosti úloh

Deklarovaná náročnost úloh je nastavena v závislosti na jednotlivých lekcích a postupně se zvyšuje. Úlohy jsou navrženy tak, aby reflektovaly stoupající úroveň komplexity – od základního seznámení s prostředím až po složité programové struktury.

- 1 – Nízká náročnost: úlohy zahrnují základní práci s příkazy, které slouží především k prvotnímu seznámení s prostředím a ovládáním želvy.
- 2 – Mírná náročnost: obsahuje úlohy pracující s cyklem, složitějším otáčením želvy, přesným zadáním úhlů a úpravou parametrů příkazů.
- 3 – Střední náročnost: vyžaduje vyšší míru abstrakce a dekompozice, zahrnuje tvorbu složitějších útvarů s využitím cyklů.
- 4 – Vysoká náročnost: úlohy obsahují vnořené cykly a vyžadují větší schopnost dekompozice problémů. Žáci musí nad řešením skutečně přemýšlet, náhodné zkoušení příkazů je vede k neúspěchu. Úlohy zahrnují také práci s proměnnými.
- 5 – Velmi vysoká náročnost: žáci pracují s komplexními strukturami, víceúrovňovým opakováním, pokročilou manipulací s proměnnými a hlubší dekompozicí i abstrakcí problémů.

Možné potíže při řešení úloh

Možné potíže při řešení úloh v Pencil Code lze rozdělit do několika kategorií. Jedním z častých problémů může být přílišná složitost úlohy. Pokud se s tímto potýká jediný žák, učitel by měl nejprve ověřit, zda správně porozuměl zadání. V případě potřeby je možné nabídnout zjednodušenou variantu úlohy nebo se zaměřit pouze na dílčí cíle. Žáci, kteří zvládnou úlohu rychleji, mohou pod vedením učitele pomáhat ostatním. Pokud je úloha složitá pro většinu třídy, lze ji řešit ve dvojicích či malých skupinách, případně učitel zadá jen její část – například místo celé „Síť šestiúhelníků“ nechá žáky vytvořit nejprve jeden šestiúhelník a teprve poté pracovat na jeho opakování pomocí cyklu.

Naopak může nastat i situace, kdy jsou úlohy příliš jednoduché. V takovém případě je vhodné nabídnout rozšířenou variantu zadání, například přidat požadavek na kresbu vícebarevných objektů nebo složitějších vzorů. Pokud žáci rychle pochopí principy úlohy, může se také zkrátit časová dotace a přejít k dalším výzvám.

Další překážkou může být nedostatek času na vypracování úlohy. Pokud žáci nestíhají z důvodu její složitosti, je možné zadat pouze dílčí cíle nebo zjednodušenou variantu. V případě, že zdržení způsobuje pomalý zápis kódu, může učitel předem připravit část programu, aby se žáci mohli soustředit jen na jeho úpravu. Pokud se žáci věnují jiným

činnostem místo práce na úloze, je vhodné využít běžné kázeňské metody nebo zavést herní prvek, například časový limit.

Motivace žáků hraje v úlohách Pencil Code zásadní roli. Nezájem často vyplývá z pocitu nadměrné obtížnosti úlohy. Učitel by proto měl zkontrolovat porozumění zadání a případně upravit obtížnost úlohy podle dovedností žáků. Velkou motivací může být i možnost vytvoření vlastního projektu, ve kterém mohou žáci nabyté znalosti využít. Učitel by měl pravidelně zdůrazňovat, jaký pokrok žáci dělají, a ukazovat jim, co se naučili oproti předchozím hodinám. Dalším způsobem motivace je personalizace zadání, například volbou barvy či tvaru, který mají v úloze vytvořit.

Metody výuky

Výuka je postavena především na problémově orientovaném učení, při kterém žáci samostatně objevují řešení zadaných úloh. Vstupní situace často představuje konkrétní problém nebo výzvu, kterou mají žáci vyřešit pomocí programování v prostředí Pencil Code. Tímto způsobem si žáci aktivně osvojují principy informatického myšlení, především dekompozici, algoritmizaci a práci s abstrakcí.

Samostatná práce je základní formou realizace této metody. Žáci pracují individuálně vlastním tempem a podle svých schopností. Tím se podporuje rozvoj samostatnosti, zodpovědnosti za vlastní učení a schopnosti hledat a testovat různá řešení.

Doplňkově je využívána také demonstrační metoda, zejména při zavádění nových programovacích bloků nebo složitějších konstrukcí. Učitel žákům ukazuje postup na konkrétním příkladu, často formou modelování kódu na tabuli nebo promítací ploše, kde společně analyzují jednotlivé kroky a jejich logiku. Tato forma přispívá k lepšímu pochopení principů algoritmizace a umožňuje žákům propojit teoretický základ s praktickou realizací.

Součástí výuky je také průběžná reflexe, která probíhá buď individuálně (např. při kontrole výstupů), nebo ve skupinové diskuzi. Žáci sdílejí různé přístupy k řešení a učí se navzájem. Díky tomu se posiluje metakognice a schopnost formulovat vlastní postup.

Žákovské projekty jsou pravidelně ukládány adresářů, což podporuje sledování pokroku, práci s verzemi kódu a možnost navazovat na dřívější práci v pozdějších lekcích.

Časová dotace

Celý časový plán lekcí je navržen tak, aby každá lekce odpovídala jedné vyučovací jednotce (45 minut). V této časové dotaci je zahrnut nejen samotný čas na řešení úloh, ale také prostor pro výklad učitele, rozbor úlohy a plynulý přechod mezi aktivitami. Na začátku a konci lekce se počítá s časem na zopakování předchozích znalostí nebo shrnutí nově získaných dovedností, přičemž konkrétní průběh závisí na povaze dané lekce. Představení zadání, případný rozbor úlohy a samostatná práce žáků trvá přibližně 5–10 minut, přičemž u složitějších úloh je časová dotace na analýzu zadání a diskusi delší. V rámci první lekce se počítá s časem na registraci do prostředí Pencil Code a úvodním seznámením se s jeho rozhraním, což může mírně prodloužit úvodní fázi první lekce. Druhá lekce navazuje na první a zahrnuje prostor pro zopakování klíčových dovedností z předchozí lekce. Pokud mezi lekcemi není bezprostřední návaznost (například vyučovací jednotky jako dvouhodinovky), může být na začátku zařazena krátká úloha na opakování. Opět se uvažuje časová dotace 5–10 minut na jednotlivé úlohy, přičemž jejich skutečné trvání se odvíjí od tempa a schopností žáků. Třetí lekce rovněž začíná opakováním klíčových konceptů a postupně přechází k náročnějším úlohám. Pro zajištění dostatečného prostoru na závěrečný projekt je doporučeno část této lekce rozdělit tak, aby přesah do čtvrté vyučovací hodiny umožňoval rezervu na finální tvorbu závěrečného projektu.

7.2 Výzkumný soubor

Výzkumné šetření bylo realizováno na jedné ze základních škol v Praze, která poskytuje vzdělávání na prvním i druhém stupni. Školu aktuálně navštěvuje přes 500 žáků a dlouhodobě se profiluje jako instituce s rozšířenou výukou jazyků a moderními technologiemi. Informatika se zde vyučuje již od 4. ročníku a škola je vybavena dvěma počítačovými učebnami, interaktivními tabulemi a dalšími digitálními pomůckami. Výuka „nové informatiky“ byla zavedena od školního roku 2021/2022.

Moje výzkumné šetření probíhalo v obou třídách 7. ročníku vybrané základní školy. Na výuku informatiky jsou tyto třídy děleny na dvě skupiny. Každá skupina byla tvořena přibližně 15 žáky. V první skupině bylo několik žáků se zvýšenou motivací a zájmem o výuku. Ve druhé skupině bylo pár žáků s OMJ-odlišným mateřským jazykem a sníženým zájmem o výuku. Druhá skupina má výuku informatiky jako odpolední vyučování. Třetí skupina byla výkonnostně vyrovnaná a pracovala stejným tempem. Ve čtvrté skupině se nacházeli dva žáci, kteří navštěvují kroužek IT na škole, jinak byla tato skupina průměrná. Různorodé složení skupin se ukázalo jako přínosné, jelikož umožnilo získat širokou škálu zpětné vazby.

Úlohy v lekcích byly navrženy jako praktická část výuky v rámci čtyř po sobě jdoucích vyučovacích hodin, zaměřených na tematický celek „Algoritmizace a programování“. Úlohy byly otestovány na obou skupinách a byly navrženy vhodně. Nabízely širší výběr a umožňovaly diferencovaný přístup k výuce.

Skupiny žáků 7. třídy ZŠ

Tito žáci již měli zkušenosti s programováním z předchozích ročníků. Pracovali ve Scratchi, později se seznámili i s Pythonem. Díky výuce blokových jazyků, například u LEGO Mindstorms, pro ně nebylo obtížné přejít na nové prostředí. S touto průpravou si snadno osvojili práci v Pencil Code a rychle se adaptovali na jeho možnosti.

Skupina 1

Tato skupina byla složena z velmi motivovaných a rychlých žáků, kteří měli vysoký zájem o informatiku a programování. Pracovali samostatně, často si úlohy rozšiřovali a aktivně se zapojovali do diskuzí.

Skupina 2

V této skupině bylo tempo pomalejší, jelikož výuka probíhala odpoledne, kdy už byli žáci unavenější. Někteří měli odlišný mateřský jazyk (OMJ), což znamenalo, že potřebovali více podpory při porozumění zadání. Přesto se s pomocí dokázali zapojit a úlohy úspěšně plnili.

Skupina 3

Tato skupina byla výkonnostně vyrovnaná, bez výrazných rozdílů mezi jednotlivými žáky. Pracovali stejným tempem, nepotřebovali tolik individuální pomoci a dokázali si navzájem poradit. Atmosféra ve skupině byla spíše neutrální, bez extrémního nadšení i nezájmu.

Skupina 4

V této skupině byly průměrné výsledky, ale zajímavé složení – výraznou část tvořily dívky, které informatika příliš nebavila. Na druhou stranu ve skupině byli dva žáci, kteří navštěvovali IT kroužek a vynikali v programování, což pomáhalo udržet ve skupině určitou dynamiku a motivaci.

8 Lekce 1 Základy

V této lekci se žáci seznámí s prostředím Pencil Code. Naučí se zakládat vlastní účet, vytvářet programy a organizovat je ve svém adresáři. Osvojí si správné pojmenovávání a ukládání programů a pochopí, jak přecházet mezi hlavním adresářem a jednotlivými programy. Dále se naučí pracovat s kategoriemi příkazů, přetahovat bloky a vnímat souvislosti mezi vytvořeným scénářem a výsledným výstupem. Lekce obsahuje čtyři úlohy, přičemž poslední úlohy – nakreslení čtverce a nakreslení slunce – plynule navazují na další lekci věnovanou cyklům.

Odkaz na Lekci: https://gawkarol.pencilcode.net/edit/1/

8.1 Doporučená realizace úloh ve výukové lekci

Úvod (5-10 minut)

- Učitel představí prostředí Pencil Code – vysvětlí základní funkce, možnosti a způsob práce v tomto prostředí.
- Žáci si zakládají vlastní účet a učí se orientovat v uživatelském rozhraní.
- Učitel ukáže, jak správně pojmenovávat a ukládat programy, jak pracovat se soubory a jak přecházet mezi hlavním adresářem a jednotlivými programy.

Samostatná práce na úlohách (20-35 minut)

Učitel vždy nejprve ukáže zadání úlohy – buď ve formě výstřižku, nebo na tabuli. Pokud je úloha složitější, učitel ji stručně rozebere a vysvětlí klíčové kroky řešení. Žáci pracují samostatně.

- Úloha Čáry Máry: žáci se učí přetahovat bloky a vnímat souvislosti mezi vytvořeným scénářem a výsledným výstupem.
- Úloha Hřib: žáci si zkouší první jednoduchý obrázek a procvičují základní pohyb želvy.
- Úloha Písmeno M: žáci procvičují základní ovládání obrazce želvy a přesného zadávání úhlů.
- Úloha Čtverec: žáci se seznámí s opakováním příkazů, které plynule vede k další lekci.

- Úloha Slunce: žáci vytvářejí opakující se vzory, což je připravuje na práci s cykly v další lekci.

Reflexe a závěr lekce (5–10 minut)

- Shrnutí naučených dovedností: učitel s žáky zopakuje klíčové poznatky (ukládání programů, základní příkazy, přetahování bloků).
- Diskuze o řešeních úloh: jaké postupy žáci zvolili, co se jim podařilo, kde měli potíže.
- Přechod k další lekci: učitel představí téma cyklů, na které budou navazovat úlohy z této lekce (čtverec, slunce).

8.2 Vstupní požadavky

Žáci mohou mít předchozí zkušenosti se Scratchem, což jim usnadní pochopení některých konceptů, ale není to nezbytné. Úlohy v Lekcích jsou koncipovány tak, aby byly přístupné i úplným začátečníkům. Každá lekce staví na předchozích znalostech a vede studenty k postupnému rozvoji informatického myšlení a praktických dovedností.

8.3 Osvojení příkazů

- „**`fd`**“: žáci se seznámí s příkazem, který budou používat velmi často; zjistí jeho funkci, úskalí (posouvá pouze dopředu) a díky čtvercové mřížce na pozadí získají představu o délce jednotky.
- „**`bk`**“: žáci se seznámí s příkazem pro zpátečku (posun dozadu).
- „**`rt/lt`**“: žáci zjistí, jak lze otáčet postavy v prostředí Pencil Code; uvědomí si, že otáčení doleva/doprava o stupěň jim umožní otáčet postavu želvy jakýkoliv směrem.
- „**`rt/lt`** o poloměr **`r`**“: žáci začnou používat příkaz pro tvorbu kružnice.
- „**`speed/speed infinity`**“: žáci si osvojí příkaz pro zrychlení pohybu obrazce želvy.
- **„`pen`“**: žáci se seznámí s příkazem pro nastavení barvy a tloušťky pera, kterou postava želvy kreslí; pochopí, že změna těchto parametrů ovlivňuje vzhled kreslených objektů.

- **„`fill`“**: žáci zjistí, jak vyplnit uzavřené oblasti barvou; naučí se správně kombinovat tento příkaz s jinými kreslicími příkazy pro dosažení požadovaného efektu.
- **„`dot`“**: žáci objeví způsob, jak nakreslit tečku určité velikosti a barvy na aktuální pozici postavy želvy; vyzkouší si, jak lze tímto příkazem vytvářet jednoduché vzory.
- **„`box`“**: žáci objeví způsob, jak nakreslit čtvereček určité velikosti a barvy na aktuální pozici postavy želvy; vyzkouší si, jak lze tímto příkazem vytvářet jednoduché vzory.

8.4 Úloha 1: Čáry Máry

8.4.1 Výukové cíle

- Žák se seznámí s uživatelským prostředím Pencil Code.
- Žák zvládá základní manipulaci s příkazy a spouštění kódu.
- Žák pochopí princip pohybu želvy a jejího kreslení.

8.4.2 Zadání

Vyzkoušej si práci v prostředí Pencil Code. Tvým úkolem je vytvořit jednoduchý obrazec pomocí pohybu želvy. Použij základní příkazy pro kreslení čar a experimentuj s různými ovládacími prvky želvy.

8.4.3 Pokyn pro žáka

Tvým úkolem je naučit se pracovat s příkazy pro pohyb a kreslení. Přetáhni do programu příkazy jako „**`fd`**" (dopředu), „**`rt/lt`**" (otočení vpravo/vlevo) a „**`pen`**" (kreslení). Po spuštění kódu sleduj, jak se želva pohybuje. Zkus měnit hodnoty v příkazech, abys viděl/a, jaký to má vliv na výsledný obrazec. Pro další odzkoušení prostředí využij i příkazy pro tvorbu tečky „**`dot`**“ nebo čtverečku „**`box`**“ z kategorie **`art`**.

V této úloze nejde prvotně o správné řešení, ale o vyzkoušení možností Pencil Code. Můžeš si hrát s různými kombinacemi příkazů a pozorovat, jak se mění výsledný tvar.

8.4.4 Didaktický význam úlohy

Tato úloha slouží jako úvodní seznámení s prostředím Pencil Code a jeho základními funkcemi. Žáci si osvojí princip přetahování příkazů, jejich úpravu a spouštění kódu.

Klíčovým prvkem je pochopení vztahu mezi sekvencí příkazů a výsledným výstupem. Úloha zároveň podporuje experimentování a rozvíjí informatické myšlení tím, že žákům umožňuje volně zkoumat možnosti prostředí.

8.4.5 Možné varianty úlohy podle náročnosti

- Základní: použití několika příkazů pro vytvoření jednoduchého obrazce.
- Rozšířená: přidání změny barvy a tloušťky čáry či vytvoření kružnice. Přidání příkazů pro tvorbu tečky a čtverečku.
- Zjednodušená: práce s předpřipravenými příkazy a jejich úpravou podle pokynů.

8.4.6 Možné obtíže a jejich řešení

- Žáci neví, jak začít: učitel může ukázat základní příkazy a vysvětlit jejich funkci.
- Nečekaný pohyb želvy: žáci si musí uvědomit, že želva vykonává příkazy postupně a každá změna ovlivní výsledek. Doporučuje se zkoušet menší úpravy kódu a sledovat jejich efekt.
- Zapomenutý příkaz pro kreslení: pokud želva nekreslí, je potřeba zkontrolovat, zda byl použit příkaz „**`pen`**".

8.5 Úloha 2: Hřib

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/1/Hrib

8.5.1 Výukové cíle

- Žák vysvětlí, proč je důležité pořadí kroků v algoritmu.
- Žák zvládá orientaci a práci v prostředí Pencil Code.
- Žák vytvoří algoritmus pro jednoduchý postup v jazyce Pencil Code.

8.5.2 Zadání

V prostředí Pencil Code nakresli hřib. Z příkazů lze vytvořit různé druhy hřibů, záleží na každém, jak velký, barevný a tlustý hříbek bude.

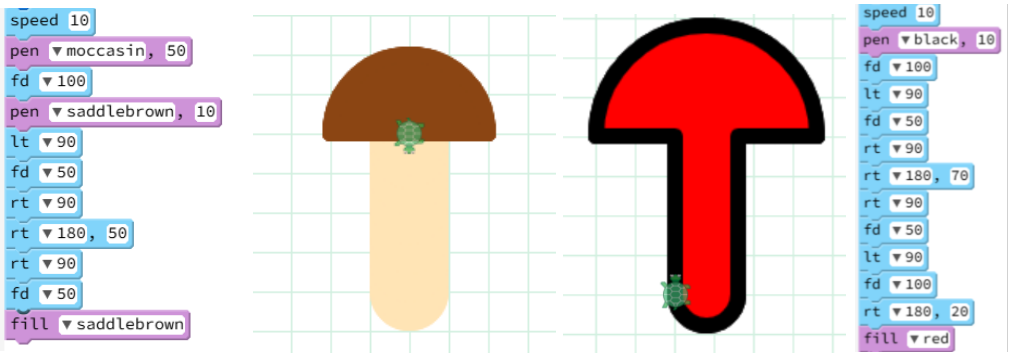
8.5.3 Pokyn pro žáka

Je potřeba si uvědomit, že výsledný obrázek hřibu se skládá ze dvou částí, a to z nožičky a kloboučku. Postavě želvy je potřeba nejdříve říct, čím a jak má kreslit. Použij příkaz „**`pen`**“ pro nadefinování barvy a tloušťky pera. Dále ji musíš říct, kam přesně má jít, tedy o kolik

kroků se musí posunout dopředu. Využij příkaz „**fd**“ případně „**bk**“. Pokud chceš obrazec želvy otáčet je třeba přetáhnout příkaz „**rt/lt**“. Při tvorbě kloboučku se musí využít příkaz pro tvorbu kružnice a to „**rt/lt**“ s poloměrem otáčení „**r**“. Pro vybarvení kloboučku barvou se použije příkaz „**fill**“.

Nezapomeň, že postavu želvy programuješ. Tvoříš pro ni postup, aby věděla, co má dělat. Každý příkaz musíš přenést z banky příkazů do scénáře a následně spustit program. Zde není zas tak důležité, zda začneme kloboučkem nebo nožičkou, ale je potřeba si uvědomit, že třeba příkaz pro vybarvení obrazce nelze použít, když obrazec není uzavřen.

8.5.4 Ukázka možného řešení z řad žáků:



Obrázek 15—Ukázka řešení pro úlohu Hřib

8.5.5 Didaktický význam úlohy

Tato úloha napomáhá rozvoji algoritmizace a seznamuje žáky s automatizací. Je důležitá pro pochopení základních principů práce v prostředí Pencil Code a řešení úloh krokováním. Žáci si osvojí práci s kombinací příkazů pro pohyb a kreslení, což jim pomůže při složitějších úlohách, například při tvorbě opakujících se vzorů pomocí cyklů v dalších lekcích. Zároveň úloha podporuje kreativitu, protože umožňuje žákům upravit vzhled hřibu podle vlastní fantazie.

8.5.6 Možné varianty úlohy podle náročnosti

- Základní: vytvoření kódu pro tvorbu obrazce hřib.
- Rozšířená: úprava barev a tloušťky pera, úprava velikosti hřibu.
- Zjednodušená: používání předdefinovaných hodnot příkazů, případně pomoc.

8.5.7 Možné obtíže a jejich řešení

- Nesprávné pořadí příkazů: zdůraznit, že fill nelze použít, pokud není tvar uzavřen. Doporučit, aby si žáci postup nejprve naplánovali.
- Problémy s otočením: pomůže vizuální znázornění úhlů na tabuli nebo papíře.
- Chyby při kreslení kružnice (půlkružnice): vysvětlit princip otočení s poloměrem a nechat žáky experimentovat s různými hodnotami.

8.5.8 Reflexe úlohy

Skupina 1.

Tato skupina zvládla úlohu velmi rychle a někteří žáci si ji sami rozšířili o další prvky, jako bylo vybarvování klobouku hřibu různými odstíny nebo experimentování s tloušťkou čáry. Díky vysoké motivaci a samostatnosti nepotřebovali téměř žádnou podporu. Všichni žáci program pojmenovali a uložili do svého adresáře.

Skupina 2.

Všichni žáci úlohu zvládli, ale práce probíhala pomaleji než u první skupiny. Někteří měli větší potíže s pochopením prvotního otáčení obrazce želvy, a proto bylo nutné nejprve na tabuli znázornit, jak se postava želvy otáčí. Po této vizuální pomůcce všichni žáci úlohu dokončili, správně program pojmenovali a uložili do svého adresáře.

Skupina 3.

Všichni žáci úlohu zvládli. Někteří žáci měli problém s prvotním otáčením obrazce želvy. Úloha jim kvůli tomu trvala déle. Všichni žáci program pojmenovali a uložili do svého adresáře.

Skupina 4.

V této skupině byla práce rozložena rovnoměrně, i když tempo bylo mírně pomalejší. Většina žáků úlohu zvládla bez větších obtíží, ale bylo patrné, že některé žákyně měly menší zájem o experimentování s rozšířením úlohy. Naopak dva žáci, kteří navštěvují IT kroužek, zvládli úlohu velmi rychle, ale trpělivě čekali na ostatní. Všichni žáci úlohu správně pojmenovali a uložili do svého adresáře.

8.5.9 Závěr úlohy

Žáci si při této úloze mohli vyzkoušet svou kreativitu a experimentovat s tvarem i barvami hřibu. Někteří se soustředili na co nejvěrnější zobrazení, jiní si s návrhem pohráli a vytvořili hřiby v různých barvách a velikostech. Při práci si uvědomili důležitost správného pořadí příkazů, zejména při kreslení a vybarvování kloboučku. Tato úloha napomáhá rozvoji algoritmizace a seznamuje žáky s automatizací, což je důležité pro pochopení základních principů práce v prostředí Pencil Code a řešení úloh krokováním. Žáci si zároveň osvojili práci s příkazy pro pohyb, otáčení a kreslení, což jim usnadní řešení složitějších úloh v dalších lekcích.

8.6 Úloha 3: Písmeno M

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/1/M

8.6.1 Výukové cíle

- Žák vysvětlí, proč je důležité pořadí kroků v algoritmu.
- Žák používá příkazy z kategorií **`art`** a **`move`** pro vytvoření úlohy.
- Žák vytvoří algoritmus pro jednoduchý postup v jazyce Pencil Code.

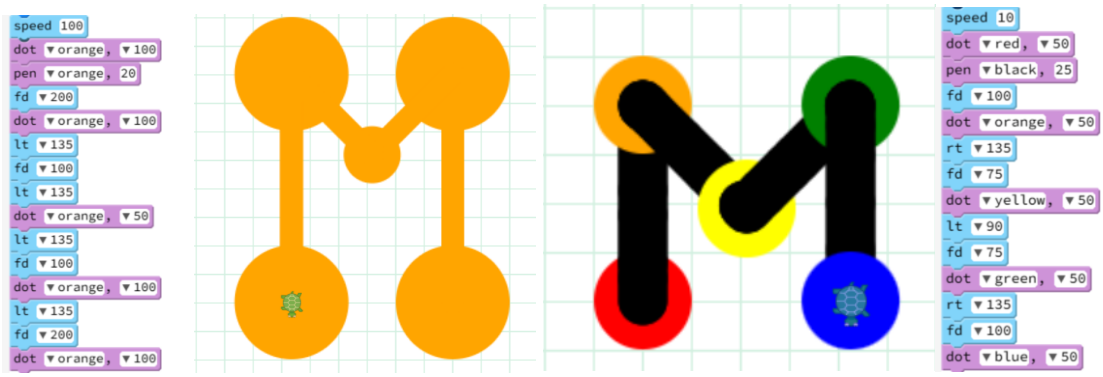
8.6.2 Zadání

V prostředí Pencil Code nakresli písmeno M a umísti tečky na jeho vrcholech. Použij různé barvy k vytvoření originálního návrhu. Můžeš začít kreslit z libovolné strany a tvar písmene přizpůsobit – může být širší nebo užší podle tvé volby.

8.6.3 Pokyn pro žáka

Při kreslení písmene M je důležité si uvědomit, že se skládá ze čtyř úseček a teček na vrcholech. Nejprve musíš postavě želvy říct, čím a jak má kreslit – k tomu použij příkaz „**`pen`**“ pro nastavení barvy a tloušťky čáry. Poté ji nasměruj správným směrem a řekni jí, kam se má posunout pomocí příkazu „**`fd`**“. Pro změnu směru využij příkazy „**`rt`**“ a „**`lt`**“ – tím zajistíš, že postava želvy vytvoří správný tvar písmene. Na vrcholy písmene umísti tečky pomocí příkazu „**`dot`**“, který vykreslí malé body na zvolených místech. Můžeš si vybrat, z jaké strany začneš kreslit, a také přizpůsobit šířku písmene. Důležité je, aby byl postup logický a vedl k požadovanému výsledku.

8.6.4 Ukázka možného řešení z řad žáků:



Obrázek 16—Ukázka řešení pro úlohu Písmeno M

8.6.5 Didaktický význam úlohy

Úloha vede žáky k pochopení souvislostí mezi kódem a výsledným výstupem. Zároveň podporuje kreativitu a logické myšlení, protože umožňuje experimentovat s různými variantami kreslení. Důraz na algoritmické myšlení pomáhá žákům připravit se na složitější úlohy, například na práci s cykly v dalších lekcích. Žáci si uvědomí, že písmeno M se skládá ze čtyř úseček, a musí si promyslet postup, jak jej nakreslit. Je třeba určit, kde začít a jaké úhly použít při otáčení.

8.6.6 Možné varianty úlohy podle náročnosti

- Základní: vytvoření kódu pro tvorbu písmena M.
- Rozšířená: úprava barev a tloušťky pera či úprava barevnosti a velikosti tečky.
- Zjednodušená: používání předdefinovaných hodnot příkazů, písmeno M bez teček, klidně nesymetricky

8.6.7 Možné obtíže a jejich řešení

- Špatné úhly otočení: žáci si mohou úhel nejprve nakreslit na papír nebo jej vizuálně znázornit na tabuli.
- Nepřesné umístění teček: vysvětlit, že tečky je třeba umístit na správné souřadnice po dokončení kreslení úseček.
- Problém s pořadím příkazů: doporučit žákům, aby si postup nejprve slovně popsali, než začnou kódovat.

8.6.8 Reflexe úlohy

Skupina 1.

Tato skupina úlohu zvládla velmi rychle a bez problémů. Někteří žáci si ji sami rozšířili tím, že zkusili experimentovat s barvami a tloušťkou čáry. Většina žáků pracovala samostatně a nebylo nutné poskytovat větší podporu. Všichni žáci úlohu správně pojmenovali a uložili do svého adresáře.

Skupina 2.

V této skupině probíhala práce pomaleji, ale všichni žáci úlohu nakonec úspěšně dokončili. Někteří žáci měli potíže s pochopením správného úhlu otočení želvy, proto bylo nutné společně na tabuli vizuálně ukázat směr pohybu. Po tomto vysvětlení byli schopni pokračovat samostatně. Všichni žáci program pojmenovali a uložili.

Skupina 3.

Žáci v této skupině pracovali vyrovnaným tempem a úlohu bez větších problémů zvládli. Někteří si potřebovali zopakovat správné pořadí příkazů, aby dosáhli vytvoření tečky na vrcholech, ale po krátkém vysvětlení už postupovali jistě. U několika žáků se objevila snaha o kreativní úpravy, například změnu barvy nebo zvětšení písmena. Všichni žáci úlohu správně pojmenovali a uložili.

Skupina 4.

V této skupině bylo tempo práce podobné jako ve třetí skupině. Některé žákyně pracovaly pomaleji, ale věnovaly úloze pozornost a snažily se ji dokončit správně. Dva žáci z IT kroužku měli úlohu hotovou velmi rychle. Všichni žáci úlohu správně pojmenovali a uložili do svého adresáře.

8.6.9 Závěr úlohy

Tato úloha byla pro žáky nejen zábavná, ale také kreativní. Během práce si mohli vyhrát s barvami, zkoušet různé tloušťky čar a přizpůsobovat tvar písmene M podle svých představ. Někteří experimentovali s umístěním teček, jiní zkoušeli různě široká či úzká písmena. Aktivita jim pomohla lépe pochopit, jak přesně ovládat postavu želvy a jak kombinovat jednotlivé příkazy k dosažení požadovaného výsledku.

8.7 Úloha 4: Čtverec

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/1/Ctverec

8.7.1 Výukové cíle

- Žák aktivně využívá příkazy z kategorií **`move`** a **`art`** prostředí Pencil Code.
- Žák dokáže kopírovat a vkládat příkazy v prostředí Pencil Code.
- Žák pochopí význam opakujících se příkazů a jejich optimalizaci pomocí cyklu.
- Žák vysvětlí, proč je důležité správné pořadí příkazů v algoritmu.
- Žák vytvoří algoritmus pro nakresleni čtverce pomocí sekvence příkazů.
- Žák následně tento algoritmus optimalizuje pomocí příkazu pro opakování.

8.7.2 Zadání

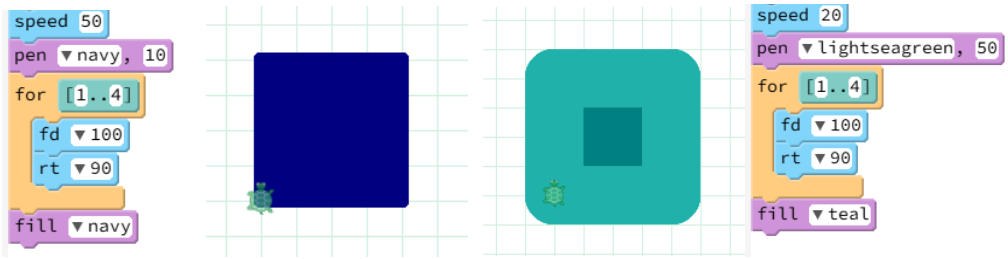
V prostředí Pencil Code nakresli čtverec. Použij základní příkazy pro pohyb a otočení a dbej na správné pořadí kroků. Experimentuj s různými barvami čáry nebo tloušťkou pera, aby byl tvůj čtverec originální. Po vytvoření čtverce sekvencí příkazů, najdi příkaz pro cyklus a optimalizuj kód.

8.7.3 Pokyn pro žáka

Čtverec se skládá ze čtyř stejných stran a čtyř otočení. Nejprve obrazci želvy nastav barvu a tloušťku čáry pomocí příkazu „**`pen`**“. Poté ji posouvej vpřed o stejnou délku pomocí příkazu „**`fd`**“ a po každém kroku ji otoč doprava o 90° pomocí příkazu „**`rt`**“. Po čtyřech krocích a čtyřech otočeních by měl být čtverec dokončen. Pro vytvoření čtverce pomocí cyklu je třeba využít příkaz z kategorie **`control`**.

Můžeš experimentovat s velikostí čtverce nebo změnit barvy a vytvořit tak zajímavější výsledek. Pokud se ti čtverec nepovede, zkontroluj, zda jsou všechny úhly a délky kroků správné.

8.7.4 Ukázka možného řešení z řad žáků:



Obrázek 17—Ukázka řešení pro úlohu Čtverec

8.7.5 Didaktický význam úlohy

Tato úloha učí žáky pracovat s příkazy pro pohyb a otáčení a zdůrazňuje důležitost správného pořadí příkazů v algoritmu. Žáci si osvojují sekvenční myšlení a propojují kód s vizuálním výsledkem. Úloha je základem pro pochopení cyklů, které budou využity v dalších lekcích k efektivnějšímu vykreslování opakujících se tvarů.

Tato úloha napomáhá rozvoji algoritmizace a automatizace a je důležitá pro pochopení základních principů práce v prostředí Pencil Code a řešení úloh krokováním.

8.7.6 Možné varianty úlohy podle náročnosti

- Základní: vytvoření čtverce pomocí sekvence příkazů „**`fd`**“ a „**`rt`**“ následná optimalizace kódu pro cyklus.
- Rozšířená: úprava barev a tloušťky pera či úprava barevnosti a velikosti tečky.
- Zjednodušená: používání předdefinovaných hodnot příkazů, čtverec pouze o dvou stranách.

8.7.7 Možné obtíže a jejich řešení

- Chybný úhel otočení: žáci si mohou nakreslit čtverec na papír a zakreslit směr pohybu obrazce želvy.
- Nesprávná délka stran: doporučit žákům, aby se ujistili, že všechny kroky mají stejnou délku.
- Chyba v pořadí příkazů: povzbudit žáky, aby si algoritmus nejprve zapsali nebo si ho slovně popsali předtím, než začnou programovat.

8.7.8 Reflexe úlohy

Skupina 1.

Tato skupina zvládla úlohu velmi rychle. Už při prvním zápisu pomocí sekvence příkazů si sami začali všímat opakujících se částí kódu a okamžitě hledali způsob, jak jej optimalizovat pomocí cyklu. Nebylo nutné je nijak vést, sami přirozeně dospěli k závěru, že cyklus je ideální řešení. Někteří žáci si úlohu rozšířili o změnu barvy čtverce nebo experimentovali s jeho velikostí. Rychlejším žákům jsem zadala doplňkový úkol – změnit tvar na jiný mnohoúhelník, což je motivovalo k dalšímu zkoumání možností cyklu. Všichni žáci program správně pojmenovali a uložili.

Skupina 2.

Všichni žáci úlohu úspěšně dokončili, ale potřebovali více podpory při přechodu ze sekvence příkazů na použití cyklu. Bylo nutné je nejprve navést otázkou, co se v kódu opakuje, a ukázat, kde v prostředí Pencil Code najdou příkaz pro opakování. Jakmile tuto informaci získali, dokázali si s optimalizací poradit sami. Po dokončení úlohy někteří zkoušeli měnit délku stran čtverce nebo barvu čáry. Všichni žáci program správně pojmenovali a uložili.

8.7.9 Závěr úlohy

Tato úloha byla skvělým úvodem do konceptu cyklu, protože žáci mohli sami objevit jeho přínos při optimalizaci opakujících se příkazů. Přechod od sekvence příkazů k cyklu proběhl u všech úspěšně a většina žáků pochopila, proč je efektivnější použít smyčku než opakovat stejné příkazy ručně. Doplňkové úkoly, jako tvorba mnohoúhelníků nebo změna parametrů čtverce, umožnily rychlejším žákům dále rozvíjet své dovednosti a motivovaly je k experimentování. Úloha tak dobře splnila svůj účel a položila základ pro další práci s cykly v programování.

8.8 Úloha 5: Slunce

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/1/Slunce

8.8.1 Výukové cíle

- Žák aktivně využívá příkazy z kategorií **`move`** a **`art`** prostředí Pencil Code.
- Žák dokáže kopírovat a vkládat příkazy v prostředí Pencil Code.

- Žák pochopí význam opakujících se příkazů a jejich optimalizaci pomocí cyklu.
- Žák vysvětlí, proč je důležité správné pořadí příkazů v algoritmu.
- Žák vytvoří algoritmus pro nakresleni slunce pomocí sekvence příkazů.
- Žák následně tento algoritmus optimalizuje pomocí příkazu pro opakování.

8.8.2 Zadání

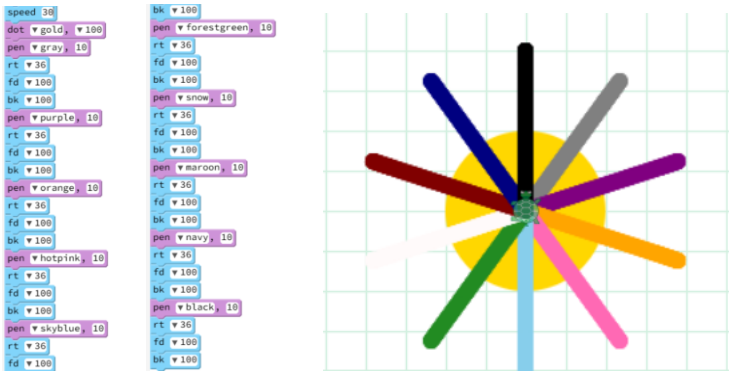
V prostředí Pencil Code nakresli slunce. Začni příkazem pro tvorbu tečky a poté vytvoř paprsky vycházející z jejího středu. Použij různé barvy pro kreativní návrh. Paprsky by měly vycházet rovnoměrně ze středu a měly by mít stejnou délku. Nezapomeň, že celkový počet paprsků a otočení o úhel by měl být dělitelný 360, protože se obrazec želvy otočí o 360 stupňů. Až úlohu dokončíš, zamysli se nad tím, zda se některé kroky v kódu neopakují. Mohla by postava želvy kreslit efektivněji?

8.8.3 Pokyn pro žáka

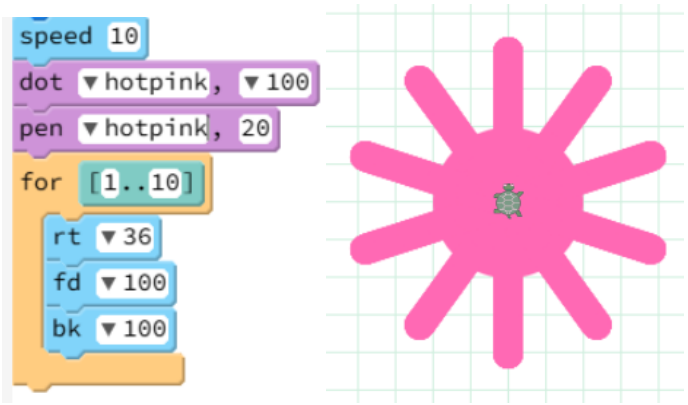
Slunce se skládá ze dvou hlavních částí: středového kruhu a paprsků. Nejprve nakresli střed pomocí příkazu „**`dot`**“, který vytvoří tečku. Poté přidej paprsky – použij příkaz „**`fd`**“ (dopředu) pro vykreslení paprsku, následně příkaz „**`bk`**“ (zpět), aby se obrazec želvy vrátila do středu. Otoč postavu želvy o stejný úhel pomocí „**`rt`**“, aby paprsky byly rovnoměrně rozmístěné. Opakuj proces pro všechny paprsky. Tím vytvoříš pravidelný sluneční vzor.

Zamysli se nad tím, zda by šlo paprsky vytvořit jednodušším způsobem. Jakým příkazem by se dalo opakování kódu zefektivnit? Najdi vhodný příkaz v kategorii „**`control`**“.

8.8.4 Ukázka možného řešení z řad žáků:



Obrázek 18—Ukázka řešení pro sekvenční zpracování příkazů pro úlohu Slunce



Obrázek 19—Ukázka řešení pro zpracování pomocí cyklu pro úlohu Slunce

8.8.5 Didaktický význam úlohy

Úloha pomáhá žákům pochopit princip opakování v algoritmech. Díky tomu si uvědomí, že místo psaní jednotlivých paprsků zvlášť mohou použít např. cyklus **`for`** pro jejich automatické generování.

Po dokončení úlohy je vedeme k reflexi nad kódem – co se v něm opakuje a jak by šlo tento opakující se vzor zjednodušit? Necháme žáky samostatně hledat řešení v kategorii „**`control`**“, kde se nachází příkazy pro opakování. Tato úloha napomáhá rozvoji algoritmizace a seznamuje žáky s automatizací – což je důležité pro pochopení základních principů práce v programování.

8.8.6 Možné varianty úlohy podle náročnosti

- Základní: žáci nakreslí s pevně daným počtem paprsků symetrické slunce. Následně zefektivní svůj kód díky cyklu. Mohou experimentovat s barvami a různými tloušťkami paprsků či mohou experimentovat s parametry v cyklu. Díky tomu vidí, jak jednoduchá změna hodnot ovlivňuje celý výsledek.
- Zjednodušená: žáci nakreslí slunce jakýmkoliv způsobem, bez použití cyklu. Každý paprsek je napsán jako samostatný příkaz. Po reflexi nad kódem objeví opakující se vzor a použijí cyklus k jeho zefektivnění.
- Rozšířená: žáci experimentují s parametry v cyklu – upravují výsledný paprsek, který může být zakončený tečkou nebo se prodloužit třeba jiným směrem. Díky tomu žáci poznají, že jediná změna v cyklu ovlivní celý výsledný obrazec a nemusí změny provádět v každém příkazu při sekvenčním zpracování.

8.8.7 Možné obtíže a jejich řešení

- Nepravidelné paprsky: vysvětlit žákům, že úhel mezi paprsky lze vypočítat jako 360 ÷ počet paprsků.
- Chyba v návratu postavy želvy: zdůraznit, že po příkazu „**`fd`**“ musí následovat příkaz „**`bk`**“, aby se postava želvy vrátila zpět.
- Problém s cyklem: ukázat, že opakování stejných kroků lze snadno zapsat pomocí cyklu **`for`**

8.8.8 Reflexe úlohy

Skupina 3.

Žáci této skupiny úlohu zvládli bez větších problémů. Nejprve pracovali se sekvencí příkazů, což jim pomohlo pochopit princip kreslení slunečních paprsků, ale kvůli většímu množství příkazů trvala tato část déle. Po zavedení cyklu si většina žáků uvědomila jeho výhody a efektivitu. Někteří rychlejší žáci zkoušeli změnit počet paprsků nebo jejich délku, což vedlo k zajímavým variantám slunce.

Skupina 4.

Ve skupině 4 se opět ukázalo, že práce v sekvenci trvá delší dobu, ale všichni žáci nakonec správně přešli na cyklus a úlohu úspěšně dokončili. Někteří žáci si museli nejprve lépe vizualizovat, jak se paprsky pravidelně rozmisťují, proto jsme si s několika z nich prošli princip rozdělení kruhu na části. Dva šikovnější žáci úlohu vyřešili velmi rychle a experimentovali s barvami a různými úhly mezi paprsky. Ostatní pracovali soustředěně a postupně úlohu dokončili.

8.8.9 Závěr úlohy

Úloha umožnila žákům pochopit rozdíl mezi sekvencí příkazů a využitím cyklu pro opakující se vzory. Přirozeně vedla k zamyšlení nad efektivitou kódu a jeho optimalizací. Díky experimentování s počtem paprsků, jejich délkou a barvou si žáci mohli úlohu přizpůsobit a lépe pochopit princip pravidelného rozmístění paprsků ve středu. Přestože práce v sekvenci zabrala více času, většina žáků si uvědomila výhody cyklu a dokázala jej správně aplikovat. Úloha byla pro všechny skupiny zvládnutelná a nabídla prostor pro kreativní rozšíření.

9 Lekce 2 Cykly

V této lekci žáci prohlubují dovednosti cyklů v prostředí Pencil Code a chápou, jak mohou opakující se sekvence příkazů zjednodušit a zefektivnit svůj kód. Na základě předchozích zkušeností si uvědomí, že některé části jejich programů obsahují opakující se vzory, a budou vedeni k jejich nahrazení cykly. Vedle základního využití cyklů se žáci naučí pracovat také s příkazem `jumpto(x,y)`, který umožňuje přesunout obrazec želvy na konkrétní souřadnice. Díky tomu budou schopni efektivněji organizovat své kresby a vytvářet složitější tvary bez nutnosti spojování čar.

Odkaz na Lekci: https://gawkarol.pencilcode.net/edit/2/

9.1 Doporučená realizace úloh ve výukové lekci

Úvod a opakování (5-10 minut)

- Učitel krátce zopakuje znalosti z předchozí lekce a zadá jinou úlohu splňující koncept první lekce pro připomenutí (může si vybrat ze čtverce, slunce nebo vločky)
- Žáci diskutují o tom, zda ve svých programech narazili na opakující se vzory příkazů.

Samostatná práce na úlohách (25–35 minut)

Učitel vždy nejprve ukáže zadání úlohy – buď ve formě výstřižku, nebo na tabuli. Pokud je úloha složitější, učitel ji stručně rozebere a vysvětlí klíčové kroky řešení. Žáci pracují samostatně.

- Úloha Schody: využití cyklu pro kreslení opakujícího se vzoru (stupně schodů) a seznámení s příkazem „`jumpto(x,y)`“
- Úloha Puzzle: vytvoření vzoru puzzle pomocí střídání přímých úseků a půlkružnic. Žáci se zamyslí, které části kódu lze zjednodušit cyklem.
- Úloha Had: vytvoření hada složeného z opakujících se teček a čtverečků.
- Úloha Kříž: kreslení kříže pomocí cyklu.

Reflexe a závěr lekce (5 minut)

- Shrnutí naučených dovedností: Co jsou cykly a jak pomáhají psát kratší kód.
- Diskuze: Jak se podařilo úlohy vyřešit, co bylo nejtěžší.

9.2 Vstupní požadavky

Žáci by měli mít osvojené základní příkazy pro pohyb a otáčení obrazce želvy (**`fd`**, **`bk`**, **`rt`**, **`lt`**), tvorbu kruhových tvarů (**`rt/lt`** o poloměr **`r`**), změnu rychlosti (**`speed/speed infinity`**), nastavení barvy a tloušťky pera (**`pen`**), vyplňování uzavřených tvarů (**`fill`**), kreslení bodů a čtverců (**`dot`**, **`box`**) a základní práci s prostředím Pencil Code.

9.3 Osvojení příkazů

- „**`jumpto`**“: žáci se seznámí díky tomuto příkazu s okamžitý přesun obrazce želvy na zadané souřadnice **(x, y)** bez kreslení.

9.4 Úloha 1: Schody

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/2/Schody

9.4.1 Výukové cíle

- Žák aktivně využívá příkazy z kategorií **`move`** a **`control`** v prostředí Pencil Code.
- Žák dokáže identifikovat opakující se vzory v sekvenci příkazů.
- Žák pochopí význam cyklů a jejich využití k optimalizaci kódu.
- Žák vysvětlí, jak cyklus usnadňuje zápis opakujících se příkazů.
- Žák vytvoří algoritmus pro nakreslení schodů pomocí sekvence příkazů.
- Žák následně tento algoritmus optimalizuje pomocí příkazu pro opakování.

9.4.2 Zadání

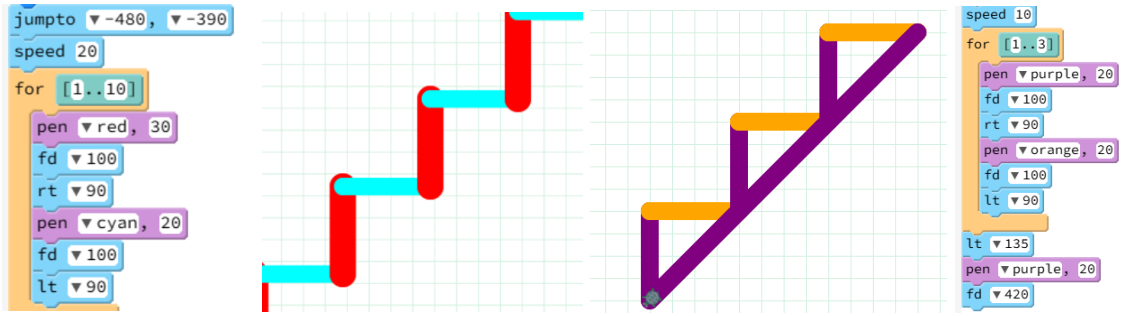
V prostředí Pencil Code naprogramuj kresbu schodů. Použij cyklus k zefektivnění programu a minimalizaci opakovaných příkazů. Schody mohou mít libovolnou velikost, barvy a počet stupňů, které si určíš.

9.4.3 Pokyn pro žáka

Při tvorbě schodů si uvědom, že každý schod se skládá ze dvou úseček – vodorovné a svislé. Pokud bychom je kreslili bez cyklu, bylo by potřeba mnoho opakovaných příkazů. Proto využij příkaz pro opakování například cyklus **`for`**, který umožní obrazci želvy vykreslit schody efektivněji.

Nejprve nastav barvu a tloušťku čáry pomocí příkazu „**`pen`“. Poté urči délku jednotlivého schodu a počet opakování. Pomocí „**`fd`“ postava želvy vykreslí vodorovnou část schodu, poté se otočí o 90° (**`rt 90`**), vykreslí svislou část pomocí dalšího příkazu „**`fd`“ a znovu se otočí zpět (**`lt 90`**), aby byla připravena na další krok. Tento postup zopakuj několikrát v cyklu. Pokud chceš schody posunout na určité místo, použij příkaz „**`jumpto`“.

9.4.4 Ukázka možného řešení z řad žáků:



Obrázek 20—Ukázka řešení pro úlohu Schody

9.4.5 Didaktický význam úlohy

Tato úloha vede žáky k rozvoji principu cyklů a jejich využití pro zjednodušení opakujících se příkazů. Žáci si uvědomí, že místo ručního vypisování jednotlivých kroků lze využít efektivnější způsob zápisu. Úloha také podporuje algoritmické myšlení, protože je nutné promyslet správné pořadí příkazů, hledání opakujících se vzorů a následně využít konečný cyklus k jejich opakování. Kromě toho se žáci učí pracovat s různými parametry příkazů, které ovlivňují výslednou podobu schodů.

9.4.6 Možné varianty úlohy podle náročnosti

- Základní: vytvoření kódu pro schody s pevným počtem stupňů.
- Rozšířená: experimentování s barvami a tloušťkou čáry tzv. dvoubarevné schody. Použití příkazu „**`jumpto`“ pro umístění schodů kamkoliv. Případně nakreslení schodů i opačným směrem.
- Zjednodušená: nakreslení schodů bez použití cyklu (pro pochopení opakování příkazů).

9.4.7 Možné obtíže a jejich řešení

- Neefektivní kód bez cyklu: pokud žáci zapisují každý schod zvlášť, vést je k zamyšlení nad opakováním a využitím příkazu pro cyklus.
- Špatné otáčení obrazce želvy: pomoci vizuálním znázorněním úhlů na tabuli nebo papíře.

9.4.8 Reflexe úlohy

Skupina 1.

Tato skupina zvládla úlohu velmi rychle a bez obtíží. Většina žáků si ihned uvědomila princip opakování a automaticky přemýšlela nad optimalizací kódu pomocí cyklu. Někteří si úlohu sami rozšířili o estetické prvky, jako změnu tloušťky čáry nebo přidání barevných prvků.

Skupina 2.

V této skupině se objevily potíže s vizualizací otáčení želvy po vykreslení každého schodu. Bylo nutné žákům ukázat správný směr otočení na tabuli. Po této pomoci již všichni úlohu úspěšně dokončili. Rychlejší žáci si vyzkoušeli změnu počet schodů.

Skupina 3.

Také v této skupině se objevily potíže s orientací a otáčením želvy po vykreslení každého stupně schodiště. Po krátkém vysvětlení a vizuální ukázce většina žáků pochopila princip a úlohu zvládla. Tempo práce bylo vyrovnané. Někteří zkusili experimentovat s velikostí schodů či s jejich počtem.

Skupina 4.

Všichni žáci úlohu zvládli, přičemž dva šikovní žáci ji měli hotovou velmi rychle. Tito žáci se snažili přidali další cyklus, který vytvářel schody směrem dolů nebo se pokusili zarovnat schody pomocí vodorovné čáry. Ostatní pracovali ve svém tempu.

9.4.9 Závěr úlohy

Úloha byla zvládnutelná pro všechny skupiny a dobře demonstrovala výhody použití cyklu při opakujících se prvcích. Pomohla žákům pochopit princip změny směru při kreslení a naučila je lépe pracovat s otáčením želvy. U některých skupin bylo potřeba vizuální

podpory, aby správně pochopily orientaci a posloupnost kroků. Kreativnější žáci využili příležitost experimentovat s dalšími cykly, což přineslo zajímavé rozšíření zadání. Úloha byla přínosná jak pro osvojení základních principů algoritmizace, tak pro rozvoj programátorského myšlení.

9.5 Úloha 2: Puzzle

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/2/Puzzle

9.5.1 Výukové cíle

- Žák aktivně využívá příkazy z kategorií **`move`**, **`art`** a **`control`** v prostředí Pencil Code.
- Žák dokáže nakreslit opakující se vzor kombinující přímky a křivky.
- Žák identifikuje opakující se sekvence v algoritmu.
- Žák chápe, jak lze využít cyklus k optimalizaci kódu.
- Žák vytvoří algoritmus pro nakreslení puzzle pomocí cyklu.

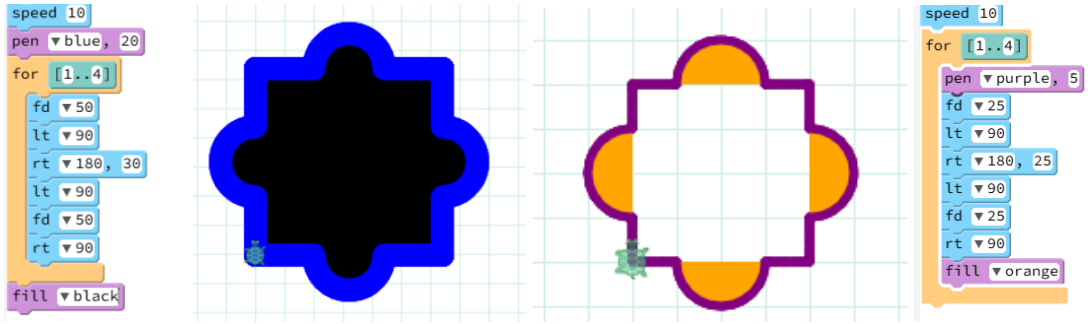
9.5.2 Zadání

V prostředí Pencil Code nakresli puzzle vzor skládající se ze střídajících se přímých úseků a půlkružnic. Použij různé barvy nebo tloušťku čáry pro zajímavější efekt a následně obrazec vybarvi. Zamysli se nad tím, zda by se některé části kódu daly zjednodušit pomocí cyklu.

9.5.3 Pokyn pro žáka

Při kreslení puzzle vzoru si uvědom, že se skládá z opakujících se úseků – rovné čáry a půlkružnic. Nejprve nastav barvu a tloušťku čáry pomocí příkazu „**`pen`**“. Poté postavu želvy nasměruj a posuň dopředu pomocí příkazu „**`fd`**“. Následně vytvoř půlkružnici pomocí příkazu „**`rt/lt`**“ o poloměr **`r`**. Dále správně zatoč obrazec želvy příkazem „**`lt`**“ a posun ji dopředu. Použij příkaz pro opakování z kategorie **`control`**. Pro vybarevní obrazce použij příkaz „**`fill`**“.

9.5.4 Ukázka možného řešení z řad žáků:



Obrázek 21—Ukázka řešení pro úlohu Puzzle

9.5.5 Didaktický význam úlohy

Úloha pomáhá žákům pochopit, jak kombinovat přímky a křivky při tvorbě opakujících se vzorů. Podporuje rozvoj algoritmického myšlení tím, že žáci musí identifikovat opakující se sekvence příkazů a následně je optimalizovat pomocí cyklu. Díky tomu si osvojí principy efektivního programování a automatizace v kódu.

9.5.6 Možné varianty úlohy podle náročnosti

- Základní: vytvoření kódu pro nakreslení puzzle vzoru pomocí cyklu.
- Rozšířená: experimentování s různými poloměry a barvama křivek. Místo půlkružnice lze vytvořit jiný opakující se tvar.
- Zjednodušená: žáci vytvoří pouze jeden segment puzzle (přímka + půlkružnice) bez použití cyklu.

9.5.7 Možné obtíže a jejich řešení

- Špatně nakreslené půlkružnice: vysvětlit, že poloměr a směr otáčení jsou klíčové, a doporučit vizuální náčrt před psaním kódu.
- Nepravidelný vzor: zdůraznit správné střídání příkazů pohybu vpřed a půlkružnic, případně vizuálně rozebrat strukturu na tabuli.
- Problém s otáčením: nechat žáky nejprve si vizualizovat, jak se želva otáčí a nakreslit bez cyklu jednu stranu puzzle. Následně zařadit do cyklu.

9.5.8 Reflexe úlohy

Skupina 3.

Na začátku úlohy někteří žáci reagovali, že je zadání obtížné. Po krátké diskuzi a rozložení úlohy na jednotlivé části si však postupně uvědomili, že se jedná o opakující se vzor. Většina žáků se zaměřila na správné vykreslení jedné části puzzle, než ji začali opakovat v cyklu. Po tomto přístupu už nebyl problém úlohu dokončit a všichni ji zvládli.

Skupina 4.

Také v této skupině byly žáci ze začátku nejistoty ohledně složitosti úlohy. Když si však žáci nakreslili jednu část puzzle a následně ji převedli do cyklu, pochopili princip a dokázali úlohu dokončit. Někteří potřebovali ještě individuální navedení, například vizuální ukázku na tabuli, jak lze jednotlivé části zopakovat. Dva žáci z kroužku IT úlohu vyřešili rychle, tak se věnovali rozšíření úlohy dle jejich uvážení.

9.5.9 Závěr úlohy

Úloha byla pro žáky ze začátku výzvou, ale po rozložení na jednotlivé části se pro ně stala pochopitelnější. Uvědomili si důležitost dekompozice problému a postupného řešení. Tento přístup jim pomohl nejen při této konkrétní úloze, ale i jako obecná strategie pro řešení složitějších problémů v programování. Použití cyklu pro opakování vzoru bylo klíčovým prvkem, který vedl k optimalizaci kódu. Celkově byla úloha přínosná a vedla k rozvoji analytického myšlení žáků. Programy byly správně pojmenovány a uloženy.

9.6 Úloha 3: Had

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/2/Had

9.6.1 Výukové cíle

- Žák aktivně využívá příkazy z kategorií **`move`**, **`art`** a **`control`** v prostředí Pencil Code.
- Žák dokáže vytvořit sekvenci opakujících se prvků (teček a čtverečků) k nakreslení hada.
- Žák chápe význam cyklu při tvorbě opakujících se vzorů.

- Žák experimentuje s barvou a velikostí prvků pro vytvoření originálního vzhledu hada.
- Žák optimalizuje kód pomocí příkazu pro opakování.

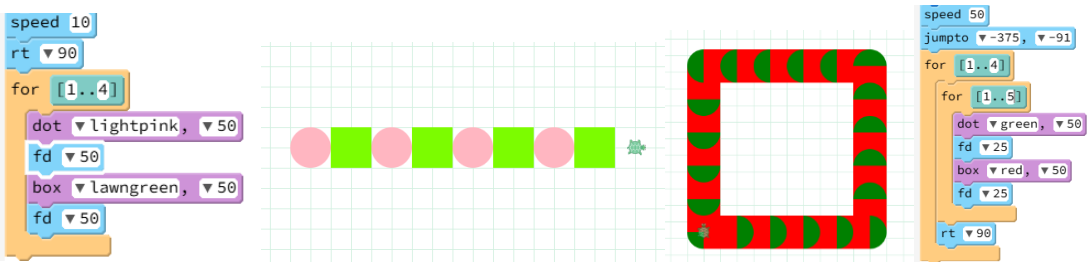
9.6.2 Zadání

V prostředí Pencil Code vytvoř vzor hada skládajícího se z opakujících se teček a čtverečků. Použij cyklus k vytvoření hada libovolné délky. Můžeš upravit barvy, velikost nebo rozestupy mezi jednotlivými segmenty hada.

9.6.3 Pokyn pro žáka

Had se skládá z opakujících se prvků – teček (**`dot`**) a čtverečků (**`box`**). Nejprve nastav barvu a velikost prvků pomocí příkazů z kategorie **`art`**. Poté pomocí cyklu vytvoř sekvenci, ve které želva postupně kreslí tečku, posune se dopředu, nakreslí čtvereček a opět se posune dopředu. Experimentuj s různými velikostmi a rozestupy mezi prvky.

9.6.4 Ukázka možného řešení z řad žáků:



Obrázek 22—Ukázka řešení pro úlohu Had

9.6.5 Didaktický význam úlohy

Úloha pomáhá žákům pochopit, jak lze využít cyklus k efektivnímu generování vzorů. Učí je pracovat s kombinací základních kreslicích příkazů (dot a box) a zdůrazňuje princip opakování. Žáci se zároveň učí přizpůsobit parametry kreslení, což podporuje jejich kreativní myšlení.

9.6.6 Možné varianty úlohy podle náročnosti

- Základní: vytvoření hada s různými barvami, velikostmi nebo náhodným umístěním prvků.
- Rozšířená: experimentování s otáčením hada či příkazem „**`jumpto`**“ pro vytvoření více hadů.

- Zjednodušená: použití pouze jednoho typu prvku (např. pouze tečky) bez úpravy.

9.6.7 Možné obtíže a jejich řešení

- Nepravidelný vzor: ujistit se, že želva se po nakreslení jednoho prvku posune stejnou vzdálenost před nakreslením dalšího.
- Problém s cyklem: pomoci žákům pochopit, že stejná sekvence příkazů se v cyklu opakuje, a nechat je nejprve napsat kód bez cyklu.
- Nevhodné rozestupy: doporučit vizuální náčrt na papír nebo tabuli, aby žáci lépe pochopili, jak se prvky hada střídají.

9.6.8 Reflexe úlohy

Skupina 1.

Tato skupina úlohu zvládla velmi rychle s čistým a efektivním kódem. Po dokončení základní verze se žáci pustili do experimentování s různými rozestupy mezi prvky hada a barvami, aby vytvořili zajímavější vizuální efekt.

Skupina 2.

Úloha žáky velmi zaujala a bavila. Překvapivě rychlí byli i žáci s OMJ, kteří úlohu pochopili téměř okamžitě a dokončili ji mezi prvními. Pravděpodobně jim pomohlo, že úkol byl vizuálně intuitivní a jasně strukturovaný. Celá skupina pracovala s nadšením a úlohu dokončila úspěšně.

Skupina 3.

Všichni žáci úlohu zvládli a několik z nich se rozhodlo experimentovat se změnou počtu prvků a velikosti jednotlivých částí hada. Tento přístup jim pomohl lépe pochopit práci s proměnnými a jejich využití v cyklech.

Skupina 4.

Tato skupina úlohu pojala kreativně. Dva kluci hada zatáčeli a někteří dokonce využili i cyklus v cyklu pro tvorbu složitějších vzorů. Holky se zaměřily na estetickou stránku a vytvořily krásné „korálkové hady“ v pastelových barvách. Úloha je bavila a většina z nich se snažila experimentovat s vizuálním vzhledem hada.

9.6.9 Závěr úlohy

Úloha byla velmi dobře přijatá napříč všemi skupinami a žáci si v ní procvičili nejen základní práci s cyklem, ale i kreativní přístup k programování. Byla vizuálně atraktivní, což usnadnilo porozumění zejména žákům s OMJ. Někteří pokročilejší žáci si sami úlohu rozšířili o další prvky, jako bylo zatáčení hada nebo využití cyklu v cyklu. Celkově byla úloha úspěšná a přínosná jak po stránce programátorské, tak po stránce tvůrčí.

9.7 Úloha 4: Kříž

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/2/Kriz

9.7.1 Výukové cíle

- Žák aktivně využívá příkazy z kategorií **`move`**, **`art`** a **`control`** v prostředí Pencil Code.
- Žák dokáže nakreslit pravidelný kříž pomocí opakujících se úseků.
- Žák pochopí princip symetrie a jeho využití v programování.
- Žák využívá cyklus k optimalizaci kódu a minimalizaci opakovaného psaní příkazů.
- Žák experimentuje s barvou, tloušťkou čáry a velikostí kříže.

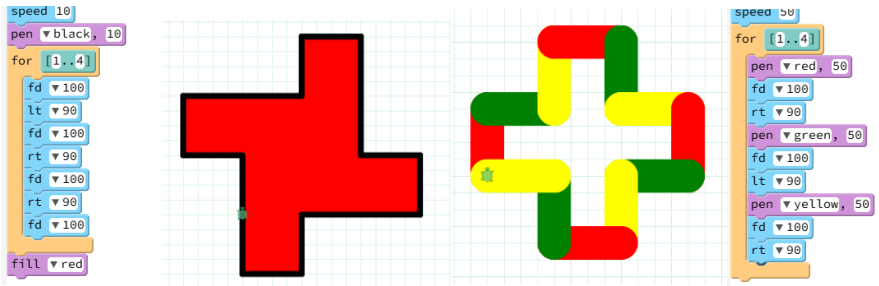
9.7.2 Zadání

V prostředí Pencil Code vytvoř kříž pomocí cyklu. Kříž se skládá z několika úseček vycházejících ze středu. Můžeš upravit jeho velikost, barvu nebo tloušťku čáry. Zamysli se, jak efektivně využít opakování, aby kód byl co nejjednodušší.

9.7.3 Pokyn pro žáka

Kříž se skládá z několika opakujících se úseček, které vycházejí ze středu. Nejprve nastav barvu a tloušťku čáry pomocí příkazu „**`pen`**“. Poté pomocí cyklu vytvoř čtyři nebo více úseček vycházejících ze stejného bodu – po každé úsečce otoč postavu želvy o pevný úhel (např. 90° pro čtyři ramena). Experimentuj s různými barvami ramen a jejich délkou.

9.7.4 Ukázka možného řešení z řad žáků:



Obrázek 23—Ukázka řešení pro úlohu Kříž

9.7.5 Didaktický význam úlohy

Úloha pomáhá žákům pochopit, jak lze využít cyklus k efektivnímu kreslení symetrických obrazců. Rozvíjí algoritmické myšlení a učí žáky pracovat se souměrností. Dále podporuje jejich kreativitu tím, že umožňuje variabilitu v návrhu kříže (různý počet ramen, délka čar, barvy).

9.7.6 Možné varianty úlohy podle náročnosti

- Základní: nakreslení jednoduchého kříže se čtyřmi rameny pomocí cyklu.
- Rozšířená: vytvoření zdobeného kříže (např. s tečkami na koncích ramen nebo s různými délkami ramen).
- Zjednodušená: nakreslení kříže bez cyklu pomocí jednotlivých příkazů pro každé rameno či nakreslení nesymetrického obrazce.

9.7.7 Možné obtíže a jejich řešení

- Chyba v cyklu: nechat žáky nejprve napsat kód pro jedno rameno a poté jej opakovat v cyklu.
- Chyba v otáčení: zdůraznit správné otáčení postavy želvy či doporučit nákres na papír.

9.7.8 Reflexe úlohy

Skupina 3.

Žáci úlohu zvládli úspěšně, i když někteří si nejprve potřebovali vizualizovat jedno rameno kříže, aby správně pochopili otáčení postavy želvy. Jakmile tento princip uchopili, pracovali

samostatně a někteří se pustili do experimentování s barvami a následného vybarvení obrazce.

Skupina 4.

Holky se opět zaměřily na vizuální stránku úlohy a vytvořily esteticky propracované kříže v různých barevných kombinacích. Kluci byli kreativní a snažili se měnit polohu či proporce kříže, což vedlo k zajímavým a jedinečným výsledkům.

9.7.9 Závěr úlohy

Úloha žákům pomohla lépe pochopit práci s otáčením obrazce želvy a využití cyklu pro tvorbu opakujících se tvarů. Zároveň podporovala kreativitu, což bylo patrné zejména u poslední skupiny, kde žáci volili různé vizuální přístupy k řešení úlohy. Aktivita byla úspěšná a žáci se u ní mohli projevit jak technicky, tak výtvarně.

10 Lekce 3 Proměnné a vnořené cykly

V této lekci se žáci naučí pracovat s vnořenými cykly a proměnnými. Seznámí se s kategorií **`operators`**, která jim umožní provádět výpočty, generovat náhodné hodnoty a pracovat s proměnnými. Důraz bude kladen na použití příkazu „**`random`**“ pro generování náhodných čísel, operátoru děleno (**`/`**) pro výpočty úhlů a správnou strukturu vnořených cyklů. Žáci tak rozšíří své dovednosti v oblasti algoritmizace a naučí se optimalizovat svůj kód pomocí proměnných.

Odkaz na Lekci: https://gawkarol.pencilcode.net/edit/3/

10.1 Doporučená realizace úloh ve výukové lekci

Úvod a opakování (5-10 minut)

- Učitel krátce zopakuje znalosti z předchozí lekce a zadá jinou úlohu splňující koncept druhé lekce pro připomenutí (může si vybrat z puzzle, kříže nebo ornament).
- Žáci diskutují o tom, zda ve svých programech narazili na opakující se vzory příkazů.

Samostatná práce na úlohách (20–35 minut)

Učitel vždy nejprve ukáže zadání úlohy – buď ve formě výstřižku, nebo na tabuli. Pokud je úloha složitější, učitel ji stručně rozebere a vysvětlí klíčové kroky řešení. Žáci pracují samostatně.

- Úloha Okno: využití vnořeného cyklu pro kreslení čtverců ve čtverci. Vylepšená verze s náhodnými barvami.
- Úloha Květina: využití proměnné pro počet okvětních lístků a jejich otočení. Experimentování s parametry.
- Úloha Kolečko z teček: využití proměnné pro x teček rovnoměrně rozmístěných po kružnici, s náhodnými barvami.

Reflexe a závěr lekce (5 minut)

- Shrnutí naučených dovedností: Jak vnořené cykly a proměnné pomáhají tvořit složitější obrazce.
- Diskuze: Co bylo nejtěžší, co by šlo udělat jinak.

- Příprava na další hodinu: „Příště budeme ještě více experimentovat s proměnnými a vytvoříme složitější vzory“

10.2 Vstupní požadavky

Žáci by měli mít osvojené základní příkazy pro pohyb a otáčení obrazce želvy („**fd**“, „**bk**“, „**rt**“, „**lt**“), tvorbu kruhových tvarů („**rt/lt**“ o poloměr **r**), změnu rychlosti („**speed/speed infinity**“), nastavení barvy a tloušťky pera („**pen**“), vyplňování uzavřených tvarů („**fill**“), kreslení teček a čtverečků („**dot**“, „**box**“) a základní práci s prostředím Pencil Code. Dále by měli být seznámeni s příkazy pro přesun obrazce želvy („**jumpto**“, „**moveto**“, „**home**“).

10.3 Osvojení příkazů

- „**random**“: žáci začnou používat příkaz pro generování náhodných hodnot, což jim umožní vytvořit dynamické a variabilní obrazce.
- „**děleno** (**/**)“: žáci si osvojí využití operátoru dělení v rámci výpočtů úhlů a jiných hodnot potřebných pro správné vykreslení obrazců.
- „**proměnná**“: žáci se naučí pracovat s proměnnými, které jim umožní ukládat a měnit hodnoty v programu, což povede k efektivnějšímu kódu.

10.4 Úloha 1: Okno

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/3/Okno

10.4.1 Výukové cíle

- Žák aktivně využívá příkazy z kategorií **move**, **art** a **control** v prostředí Pencil Code.
- Žák se naučí vnořovat cykly do sebe pro vytváření složitějších obrazců.
- Žák se naučí využívat příkazy z kategorie **operators** v prostředí Pencil Code.
- Žák pochopí princip dekompozice úlohy – rozdělení kreslení na menší opakující se části.
- Žák se seznámí s možností dynamického nastavování barev pomocí náhodné volby (**random**).

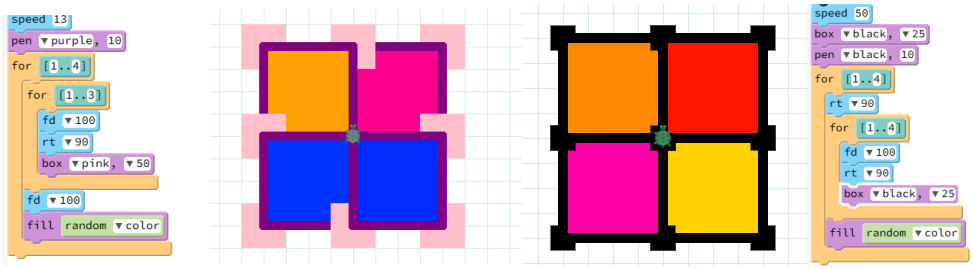
10.4.2 Zadání

V prostředí Pencil Code nakresli okno složené ze čtyř čtverců vycházejících ze stejného bodu. Použij vnořený cyklus pro nakreslení jednotlivého čtverce a hlavní cyklus pro umístění čtyř čtverců do správných pozic. V rozšířené verzi umísti do vrcholů okna čtverečky a nahraď pevné hodnoty proměnnými nebo náhodnými barvami (**`random color`**).

10.4.3 Pokyn pro žáka

Nejprve vytvoř jeden čtverec pomocí cyklu, ve kterém čtyřikrát posuneš postavu želvy dopředu a otočíš ji o 90 stupňů. Poté tento kód vlož do dalšího cyklu, aby se čtverce opakovaly čtyřikrát a vytvořily tak celé okno. Použij souřadnice nebo posuny, aby se jednotlivé čtverce správně umístily. Po dokončení hlavní části přidej do vrcholů okna tečky pomocí příkazu „**`box`**“. Nakonec svůj kód rozšiř tím, že pevné hodnoty nahradíš proměnnými a použiješ náhodné barvy k vyplnění jednotlivých čtverců.

10.4.4 Ukázka možného řešení z řad žáků:



Obrázek 24—Ukázka řešení pro úlohu Okno

10.4.5 Didaktický význam úlohy

Tato úloha pomáhá žákům pochopit princip vnořených cyklů a jejich využití v algoritmizaci. Žáci si uvědomí, jak lze složité obrazce rozložit na jednodušší části a postupně je skládat. Práce s proměnnými a náhodnými hodnotami podporuje kreativitu a rozvíjí pochopení programátorských konceptů.

10.4.6 Možné varianty úlohy podle náročnosti

- Základní: nakreslení okna pomocí vnořeného cyklu.
- Rozšířená: přidání čtverečků ve vrcholech, práce s proměnnými a náhodnými barvami.

- Zjednodušená: použití pevně daných souřadnic a hodnot bez vnořených cyklů.

10.4.7 Možné obtíže a jejich řešení

- Nepravidelné rozmístění čtverců: vysvětlit, jak pracovat se souřadnicemi a posuny mezi čtverci.
- Chyby ve vnořených cyklech: doporučit postupné budování kódu a vizuální rozbor na tabuli.
- Problémy s barvami a proměnnými: vysvětlit, jak správně generovat a přiřazovat hodnoty.

10.4.8 Reflexe úlohy

Skupina 1.

Tato skupina přivítala složitější obrazec s nadšením a úlohu splnila bez problémů. Někteří žáci se do programování pustili rovnou, ale vzhledem k první složitější úloze jsem zdůraznila důležitost dekompozice, aby si uvědomili, jak lze úlohu rozložit na menší části a efektivně ji naprogramovat.

Skupina 2.

Úloha jim trvala déle, proto jsme se na začátku více věnovali jejímu rozboru a vizualizaci. I když postupovali pomaleji, všichni úlohu dokončili a pochopili princip vnořeného cyklu.

Skupina 3.

Žáci pracovali podobným tempem, jak je pro tuto skupinu typické. Úlohu dokončili, ale zabralo jim to o něco více času než například skupině 1 či 4.

Skupina 4.

Tato skupina tradičně ráda experimentuje s barvami a tvary, což se projevilo i zde. Žáci si velmi oblíbili příkaz „`random color`“ a s nadšením jej využívali. Klukům jsem dala možnost pracovat na vlastním obrázku vedle hlavního zadání, pokud budou držet tempo s úlohami, což se jim osvědčilo jako motivace.

10.4.9 Závěr úlohy

Všem skupinám bylo na začátku zdůrazněno, jak si úlohu rozdělit na menší části a postupně ji programovat. Dále jsem předvedla nový příkaz „**`random color`**“, který si žáci okamžitě oblíbili a ve většině případů pak při dalších úlohách už nepoužívali žádné jiné barvy. Během lekce jsem si všimla, že u některých žáků postupně opadá pozornost při pojmenovávání programu – někteří si ho pojmenovali jinak, jiní ho nechali výchozí „untitled“.

10.5 Úloha 2: Květina

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/3/Kvetina

10.5.1 Výukové cíle

- Žák pochopí princip opakování kreslení kruhů pomocí cyklu.
- Žák se naučí používat proměnné k definování vlastností obrazce.
- Žák vytvoří algoritmus pro nakreslení květiny složené z pěti kružnic.
- Žák využije proměnné a náhodné hodnoty pro dynamickou tvorbu barev a vzorů.

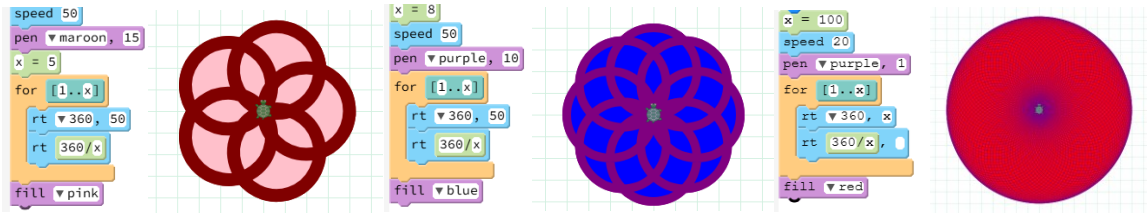
10.5.2 Zadání

V prostředí Pencil Code nakresli kytku složenou z pěti kružnic. Každá kružnice bude představovat jeden okvětní lístek. Použij cyklus k jejich nakreslení. Následně pro svůj kód nastav proměnnou pro počet okvětních lístků a implementuj ji do cyklu a do úhlu otočení mezi jednotlivými kružnicemi. Experimentuj s různými hodnotami, aby se obrazec stal zajímavějším.

10.5.3 Pokyn pro žáka

Nejprve nastav barvu a tloušťku čáry pomocí příkazu „**`pen`**“. Poté vytvoř jeden kruhový lístek pomocí příkazu „**`rt/lt`**“ s poloměr „**`r`**“. Tento proces zopakuj pětkrát v rámci cyklu a po každém kroku otoč želvu o stejný úhel, aby se okvětní lístky rovnoměrně rozmístily kolem středu. Použij proměnnou „**`x`**“ pro definování počtu okvětních lístků (kružnic) a implementuj ji do cyklu a do úhlu pro otočení.

10.5.4 Ukázka možného řešení z řad žáků:



Obrázek 25—Ukázka řešení pro úlohu Květina

10.5.5 Didaktický význam úlohy

Tato úloha rozvíjí pochopení cyklů a práce s proměnnými. Žáci se učí efektivně používat opakování, aby vytvořili složitější obrazce. Práce s proměnnými pomáhá pochopit, jak lze flexibilně měnit parametry bez nutnosti přepisovat celý kód.

10.5.6 Možné varianty úlohy podle náročnosti

- Základní: nakreslení květiny složené z pěti kružnic pomocí cyklu a implementování proměnné.
- Rozšířená: experimentování s počtem kružnic, různými tvary nebo barvami.
- Zjednodušená: použití pevně daných hodnot bez proměnné.

10.5.7 Možné obtíže a jejich řešení

- Nesprávné rozložení kružnic: vysvětlit, že celkový úhel (360°) se dělí podle počtu kružnic.
- Chybné použití proměnné: ukázat, jak proměnná ovlivňuje výpočet úhlu a umožňuje snadnou změnu kódu.

10.5.8 Reflexe úlohy

Skupina 1.

Tato skupina úlohu zvládla velmi rychle, a proto jsme se hned poté zaměřili na definování a implementaci proměnné. Úloha sloužila jako první vhled do práce s proměnnými, což všichni pochopili a úlohu úspěšně optimalizovali. Někteří žáci se dokonce pustili do úpravy svého kódu a proměnných, čímž vytvořili zajímavé a originální obrazce, které jsem však nezahrnovala do ukázek řešení.

Skupina 2.

Žáci si hned uvědomili, že kružnice budou tvořit pomocí cyklu, a tak jsem je nechala úlohu naprogramovat samostatně bez zásahu. Všichni úlohu dokončili bez problémů. Poté jsme se zaměřili na aplikování proměnné a vysvětlili jsme si, proč je její použití důležité pro budoucí úlohy.

Skupina 3.

Úlohu zvládli všichni a jako obvykle pracovali podobným tempem. Jelikož jsme se v této úloze soustředili hlavně na proměnnou, vysvětlili jsme její význam a praktické použití. Někteří žáci se ptali, proč přidávat proměnnou, když mohou hodnoty měnit ručně. Na to jsme si ukázali různé situace (z mého osobního adresáře), kdy se proměnná hodí – například při změně více hodnot najednou nebo při snazší úpravě kódu.

Skupina 4.

Žáci v této skupině opět vytvořili krásné variace květin. Dva kluci z IT kroužku ihned pochopili princip proměnných a bez dalšího vysvětlování je začali implementovat do svého kódu v Pencil Code. Dokonce experimentovali s větším počtem kružnic a vytvořili krásné mandaly. Ostatním jsem vysvětlovala postup krok za krokem pomocí ukázek, což pomohlo všem k úspěšnému dokončení úlohy.

10.5.9 Závěr úlohy

Všichni žáci optimalizovali svůj kód pomocí proměnné a vyzkoušeli si její použití. Tato úloha byla jejich prvním praktickým setkáním s proměnnými, což vedlo k zajímavým experimentům a různým variacím květin.

10.6 Úloha 3: Kolečko z teček

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/3/Kolecko

10.6.1 Výukové cíle

- Žák využívá cyklus k opakovanému vykreslení bodů v kruhovém vzoru.
- Žák pochopí, jak rozdělit kruh na rovnoměrné části a jak určit správné úhly otáčení.
- Žák pracuje s proměnnou.
- Žák experimentuje s různými parametry (počet teček, jejich rozmístění, barvy).

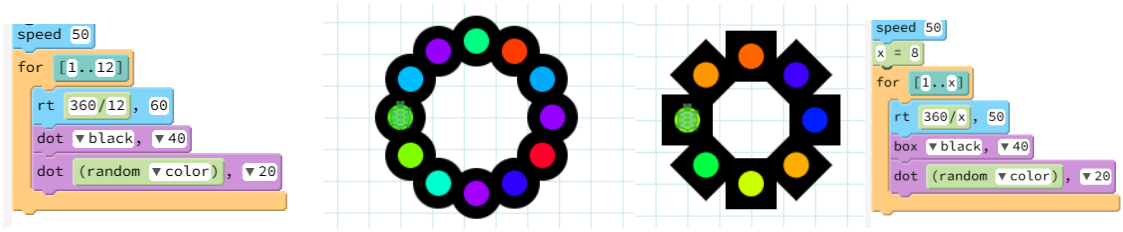
10.6.2 Zadání

V prostředí Pencil Code vytvoř kolečko složené z 12 teček. Použij cyklus k jejich rovnoměrnému rozmístění po kružnici a proměnnou **`random`** pro změnu barvy každé tečky. Experimentuj s počtem bodů, velikostí a vzdáleností mezi nimi, aby vznikly zajímavé obrazce.

10.6.3 Pokyn pro žáka

Použij cyklus k umístění 12 teček rovnoměrně po kružnici. Každou tečku vykresli příkazem „**`dot`**“, poté se posuň vpřed a otoč se o správný úhel (360° / počet teček). Barvu každé tečky nastavuj dynamicky pomocí **`random color`**. Experimentuj s různým počtem teček a jejich rozmístěním. Zkus změnit mezery mezi tečkami, jejich velikost nebo využij příkaz pro čtvereček „**`box`**“.

10.6.4 Ukázka možného řešení z řad žáků:



Obrázek 26—Ukázka řešení pro úlohu Kolečko z tečky

10.6.5 Didaktický význam úlohy

Tato úloha pomáhá žákům pochopit rovnoměrné dělení kruhu a práci s cykly. Procvičují proměnné a náhodné hodnoty, což jim umožňuje vytvářet dynamické obrazce. Experimentování s parametry podporuje kreativní myšlení a porozumění algoritmizaci.

10.6.6 Možné varianty úlohy podle náročnosti

- Základní: vytvoření kolečka z 12 teček s pevně nastavenou velikostí a barvou v cyklu.
- Rozšířená: použití proměnné **`random`** k náhodné změně barvy a velikosti teček. Možnost experimentovat s jiným počtem teček, mezery či jiným tvarem.
- Zjednodušená: nakreslení několika teček ručně bez použití cyklu.

10.6.7 Možné obtíže a jejich řešení

- Špatné rozmístění teček: vysvětlit, že správné rozdělení kruhu je 360° / počet teček.
- Problém s orientací želvy: ukázat, že je důležité se po každé tečce otočit o správný úhel.

10.6.8 Reflexe úlohy

Skupina 1.

Tato úloha byla pro skupinu spíše oddychová. Všichni ji zvládli velmi rychle, a někteří žáci automaticky použili proměnnou. Ostatním jsem ji připomněla a pobídla je, aby úlohu optimalizovali tímto způsobem.

Skupina 2.

Žáci pracovali pomalejším tempem, ale úlohu úspěšně dokončili. Místo proměnné někteří využili příkaz dělení (`/`), což bylo také funkční řešení. Práci s cykly měli dobře osvojenou, ale někteří zpočátku používali příkaz „`pen`“ k zapnutí kreslení. Postupně si uvědomili, že obrazec želvy stačí jen posouvat, a patřičně svůj kód upravili.

Skupina 3.

Očekávala jsem, že tato skupina úlohu zvládne bez problémů, což se potvrdilo. Bylo jim však potřeba zdůraznit, že podle počtu prvků v cyklu je nutné správně nastavit otáčení postavy želvy – tedy 360° děleno počtem prvků. Po této připomínce pracovali všichni správně.

Skupina 4.

Úloha se žákům líbila, především holkám, které vytvářely esteticky pěkné vzory. Kluci experimentovali s většími mezerami mezi tečkami nebo zaměnili příkaz „`dot`“ za „`box`“, čímž získali různé vizuální efekty.

10.6.9 Závěr úlohy

Všechny skupiny úlohu zvládly, a díky ní si procvičily práci s cykly a úhly otáčení. U některých bylo potřeba zdůraznit význam proměnných a efektivního kódu. Experimentování s mezerami, tvary a barvami vedlo k zajímavým variacím, což ukázalo kreativní přístup žáků.

11 Lekce 4 Složitější konstrukce

V této lekci žáci zúročí všechny dosud získané znalosti a dovednosti. Efektivně pracují s vnořenými cykly a proměnnými, což jim umožní vytvářet složitější vzory a lépe pracovat s opakující se částmi kódu. Prohlubují své schopnosti v oblasti algoritmizace a učí se optimalizovat zápis programu pomocí proměnných a dalších nástrojů, které vedou k přehlednějšímu a efektivnějšímu kódu.

Odkaz na Lekci: https://gawkarol.pencilcode.net/edit/4/

11.1 Doporučená realizace úloh ve výukové lekci

Úvod a opakování (5-10 minut)

- Učitel krátce zopakuje znalosti z předchozích lekcí a zadá úlohu na zopakování, a to mašli nebo kolečko z teček.

Samostatná práce na úlohách (20-30 minut)

Učitel vždy nejprve ukáže zadání úlohy – buď ve formě výstřižku, nebo na tabuli. Pokud je úloha složitější, učitel ji stručně rozebere a vysvětlí klíčové kroky řešení. Žáci pracují samostatně.

- Úloha Síť šestiúhelníků: vytvoření pravidelného vzoru podobného hernímu poli AZ kvízu. Použití vnořeného cyklu a proměnných.
- Úloha Hvězda: konstrukce složená z šesti rovnostranných trojúhelníků pomocí vnořených cyklů, rotace a proměnných. Barevné vyplnění pomocí příkazu „`random color`“.

Závěrečný projekt (15–20 minut)

- Samostatná nebo práce ve dvojicích na vlastním kreativním projektu.
- Učitel motivuje k originalitě například využití cyklů, proměnných, náhodných barev apod.

11.2 Vstupní požadavky

Žáci by měli mít osvojené základní příkazy pro pohyb a otáčení obrazce želvy („`fd`“, „`bk`“, „`rt`“, „`lt`“), tvorbu kruhových tvarů („`rt/lt`“ o poloměr `r`), změnu rychlosti

(„**speed/speed infinity**“), nastavení barvy a tloušťky pera („**pen**“), vyplňování uzavřených tvarů („**fill**“), kreslení teček a čtverečků („**dot**“, „**box**“) a základní práci s prostředím Pencil Code. Dále by měli být seznámeni s příkazy pro přesun obrazce želvy („**jumpto**“, „**moveto**“, „**home**“), s využitím příkazu „**random**“ pro generování náhodných hodnot a tvorbu variabilních obrazců, s použitím operátoru „**/**“ pro dělení v rámci výpočtů a s prací s proměnnými, které umožňují efektivnější a přehlednější zápis programu.

11.3 Úloha 1: Hvězda

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/4/Hvezda

11.3.1 Výukové cíle

- Žák využívá vnořený cyklus k vytvoření složitějšího geometrického obrazce.
- Žák pracuje s úhly a pochopí princip kreslení rovnostranného trojúhelníku.
- Žák experimentuje s proměnnou **random** pro nastavení barvy výplně.
- Žák si uvědomí dekompozici složitějších obrazců na menší opakující se části.

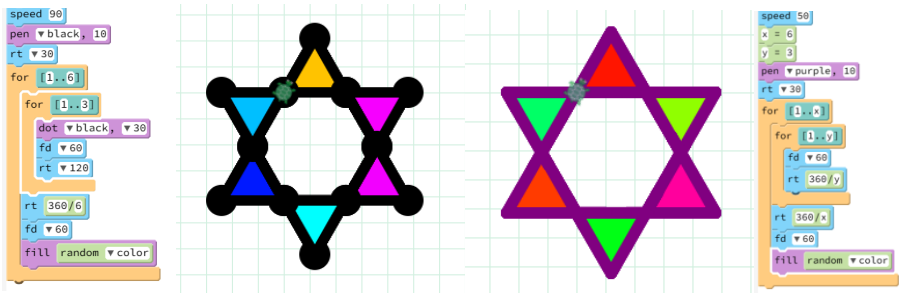
11.3.2 Zadání

V prostředí Pencil Code nakresli Davidovu hvězdu, která se skládá ze šesti rovnostranných trojúhelníků. Použij vnořený cyklus – první cyklus bude kreslit jednotlivé trojúhelníky a druhý se postará o jejich rotaci do správné polohy. k vyplnění trojúhelníků využij náhodné barvy (**random color**).

11.3.3 Pokyn pro žáka

Nejprve si rozmysli, jak nakreslit jeden rovnostranný trojúhelník – budeš potřebovat třikrát nakreslit úsečku (**fd**) a otočit se o 120°. Poté tento trojúhelník zopakuj šestkrát s otáčením o 60°, aby vznikla Davidova hvězda. Barvu každého trojúhelníku nastavuj pomocí **random color**.

11.3.4 Ukázka možného řešení z řad žáků:



Obrázek 27—Ukázka řešení pro úlohu Hvězda

11.3.5 Didaktický význam úlohy

Tato úloha učí žáky pracovat s vnořenými cykly a systematicky vytvářet složitější obrazce z opakujících se tvarů. Pomáhá jim pochopit úhly a principy symetrie. Použití proměnných a náhodných hodnot podporuje experimentování a kreativitu.

11.3.6 Možné varianty úlohy podle náročnosti

- Základní: nakreslení Davidovy hvězdy pomocí pevně daných příkazů a cyklu.
- Rozšířená: použití proměnné **`random`** pro změnu barev trojúhelníků. Možnost přidat další grafické prvky.
- Zjednodušená: nakreslení pouze jednoho rovnostranného trojúhelníku bez cyklu.

11.3.7 Možné obtíže a jejich řešení

- Nesprávně nakreslený trojúhelník: vysvětlit, že rovnostranný trojúhelník má úhly 60° a otočení želvy musí být 120°.
- Chyby při otáčení hvězdy: zdůraznit, že pro rovnoměrné rozmístění trojúhelníků je třeba otočit se vždy o 60°.
- Problém s dekompozicí celé úlohy: doporučit vizualizovat si na papír čím začít.

11.3.8 Reflexe úlohy

Skupina 1.

Tato skupina úlohu zvládla téměř samostatně. Lehce jsem naznačila princip dekompozice – že jeden cyklus vytvoří trojúhelník a druhý šest těchto trojúhelníků uspořádaných do hvězdy. Nepožadovala jsem zde nutně použití proměnné, ale někteří šikovnější žáci ji přesto

implementovali. Někteří také přidali tečky na vrcholy hvězdy pro vylepšení vizuálního dojmu.

Skupina 2.

Žáci měli větší potíže s pochopením úlohy. I přes slovní dekompozici bylo nutné znázornit klíčový moment na tabuli – konkrétně krok, kdy se želva musí vrátit a znovu obkreslit jednu stranu trojúhelníku, aby se dostala zpět do výchozí polohy. Po této vizualizaci všichni úlohu dokončili správně.

11.3.9 Závěr úlohy

Úloha byla náročnější na pochopení a vyžadovala správnou dekompozici. První skupina se s tímto principem popasovala téměř okamžitě, zatímco druhá skupina potřebovala podrobnější vizuální vysvětlení. Přesto ji všichni úspěšně dokončili a někteří ji dokonce vylepšili o další grafické prvky, jako jsou tečky na vrcholech.

11.4 Úloha 2: Síť šestiúhelníků

Odkaz na zadání úlohy: https://gawkarol.pencilcode.net/edit/4/Az

11.4.1 Výukové cíle

- Žák pochopí princip vnořeného cyklu a jeho využití při kreslení složitějších vzorů.
- Žák se naučí kreslit pravidelný šestiúhelník pomocí opakování a správných úhlů.
- Žák procvičí postupné posouvání želvy k vytvoření sítě šestiúhelníků.
- Žák uváží implementaci proměnné i zařazení příkazu `random color`.

11.4.2 Zadání

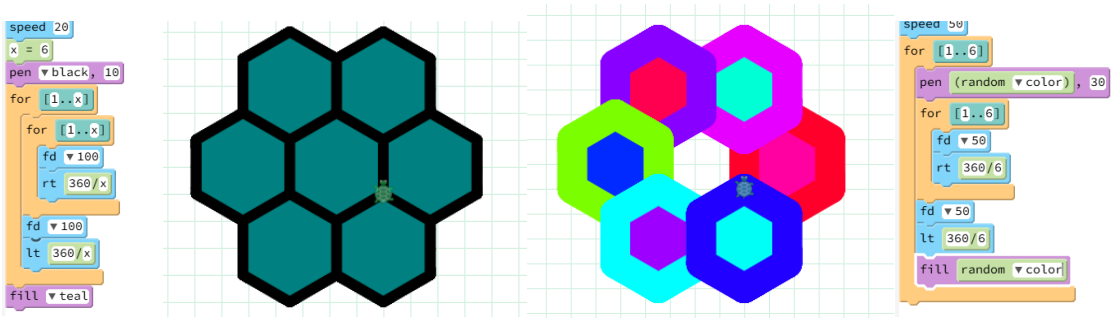
Vytvoř síť šestiúhelníků podobnou hernímu poli AZ kvízu. Každý šestiúhelník bude barevný, jak uznáš za vhodné a bude mít pravidelné uspořádání. Použij vnořený cyklus – první cyklus zajistí kreslení jednoho šestiúhelníku, zatímco druhý je rozmístí do řad. Využij proměnné.

11.4.3 Pokyn pro žáka

Nejprve nakresli jeden šestiúhelník – k tomu budeš potřebovat šestkrát vykreslit úsečku a otočit se vždy o 360/6 stupňů. Poté ho posuň kupředu o délku jeho strany a vytvoř další.

Abychom vytvořili celou síť, vložíme kreslení šestiúhelníku do vnořeného cyklu. Vybarvi síť šestiúhelníků podle libosti. Nadefinuj si proměnnou pro zefektivnění kódu.

11.4.4 Ukázka možného řešení z řad žáků:



Obrázek 28—Ukázka řešení pro úlohu Síť šestiúhelníků

11.4.5 Didaktický význam úlohy

Tato úloha umožňuje žákům procvičit princip vnořeného cyklu a pochopit, jak lze opakovaně generovat pravidelné tvary. Důležitým prvkem je také správné uspořádání tvarů a využití vyplňování barvou. Žáci se tak seznámí s efektivní strukturou kódu, která minimalizuje množství ručně psaných příkazů.

11.4.6 Možné varianty úlohy podle náročnosti

- Základní: nakreslení sítě šestiúhelníků s vnořeným cyklem a základními parametry.
- Rozšířená: použití **`random color`** k obarvení šestiúhelníků různými barvami a použití proměnné.
- Zjednodušená: použití cyklu pouze pro kreslení jednoho šestiúhelníku.

11.4.7 Možné obtíže a jejich řešení

- Chyby v úhlech šestiúhelníku: ujistit se, že želva se otáčí přesně o 360/6 = 60°.
- Problémy s rozmístěním šestiúhelníků: doporučit vizuální rozkreslení nebo přidání komentářů do kódu pro lepší pochopení posunů.
- Špatně vyplněné tvary: zajistit, že příkaz „**`fill`**“ je použito až po dokončení tvaru, jinak se barva nemusí správně aplikovat.

11.4.8 Reflexe úlohy

Skupina 3.

Tato úloha byla pro žáky náročnější, a ne všichni ji dokončili. Největší problém nastal při zatáčení obrazce želvy a pochopení, jak jednotlivé šestiúhelníky na sebe navazují. Pro některé byla abstrakce celého postupu složitá. Proto jsem nakonec na tabuli ukázala správné řešení a podrobně vysvětlila kód a logiku postupu.

Skupina 4.

Úloha byla výzvou i pro tuto skupinu a trvala delší dobu. Až na pár výjimek ji ale nakonec zvládli. Kluci se snažili implementovat proměnnou, což jim pomohlo k lepší variabilitě kódu. Celkově se jednalo o jednu z nejnáročnějších úloh v dosavadním průběhu.

11.4.9 Závěr úlohy

Úlohy „Síť šestiúhelníků“ a „Davidova hvězda“ byly pro žáky nejobtížnější. Vyžadovaly pokročilé pochopení dekompozice, správné úhly otočení a práci s opakováním. Z tohoto důvodu by bylo vhodné doporučit je spíše zkušenějším žákům. Metodicky bych je ponechala v nezměněné podobě, ale jejich zařazení do výuky by mělo být na uvážení učitele podle schopností konkrétní třídy.

11.5 Závěrečný projekt

Samostatná tvorba na závěr lekcí poskytla žákům prostor pro kreativní zpracování nabytých znalostí a dovedností. V posledních 10–20 minutách výuky, v závislosti na rychlosti jednotlivých skupin, měli možnost navrhnout a naprogramovat vlastní projekt. Pro inspiraci jim byly ukázány různé příklady z mého osobního adresáře.

Většina žáků se aktivně zapojila do tvorby vlastních obrazců a programů, přičemž hojně využívali příkaz „**`random color`**“, nekonečné cykly, ale také například příkaz „**`wear`**“ pro změnu vzhledu želvy. Někteří zabrousili i do kategorie **`sound`**, kde experimentovali se zvuky a hudebními efekty. Dívky se často soustředily na estetickou stránku výtvorů, zatímco mnozí žáci objevovali nové možnosti prostředí Pencil Code, například zkoumáním dalších kategorií příkazů. Tento přístup naznačuje, že se v prostředí cítili jistě a měli chuť experimentovat.

11.6 Shrnutí akčního výzkumu

Byl vytvořen ucelený soubor aktivit zaměřených na rozvoj informatického myšlení, který metodou akčního výzkumu ověřila v pedagogické praxi. Úlohy byly navrženy tak, aby postupně pokryly klíčové koncepty programování v prostředí Pencil Code, přičemž jejich obsah systematicky rozvíjel porozumění základním programovacím strukturám a podporoval algoritmické myšlení.

Výzkum probíhal ve čtyřech různých skupinách žáků, přičemž každá absolvovala dvě dvouhodinové výukové lekce. První část aktivit se zaměřovala na seznámení žáků s principy algoritmizace a tvorbou kódu, a to prostřednictvím úloh jako „Hřib“, „Písmeno M“, „Čtverec“ a „Slunce“. V metodických pokynech k těmto úlohám byl kladen důraz na podrobný rozbor doporučených bloků a sekvencí příkazů, aby si žáci osvojili základní strukturu programování v prostředí Pencil Code.

V další fázi výuky se žáci postupně seznamovali s konceptem cyklů a jejich aplikací v algoritmizaci. Úlohy byly navrženy tak, aby podporovaly hledání vzorů, abstrakci a efektivitu kódu. S rostoucí náročností úloh se zvyšoval důraz na vnořené cykly, práci s proměnnými a jejich význam v optimalizaci programu. Klíčovými prvky výuky se staly dekompozice problému, vizualizace a pochopení struktury kódu, což se ukázalo jako zásadní pro hlubší porozumění programovacím principům.

Žáci si nejlépe osvojili hledání vzorů, ve kterém většinou nechybovali a dokázali jej úspěšně aplikovat při řešení úloh. Největší obtíže jim však způsobovalo správné otáčení a směr želvy, stejně jako míra abstrakce a vizualizace daného tvaru. Ukázalo se, že kvalitní rozbor úlohy před samotným programováním výrazně zkracoval dobu potřebnou k jejímu dokončení hlavně u složitějších konstrukcí. Zároveň dekompozice podporovala hlubší zamyšlení nad jejím řešením. Naopak při absenci rozboru žáci často volili metodu „pokus-omyl“, kdy se snažili řešení odhadovat namísto jeho promyšleného sestavení.

Navržené úlohy nebylo nutné výrazně upravovat, díky inspiraci tvorbě úloh z portálu Umíme informatiku – Želví grafika a zároveň je koncipovala tak, aby podporovaly rozvoj informatického myšlení. Každá úloha byla promyšleně navržena s ohledem na možné obtíže, a proto byly vytvořeny i zjednodušené varianty zadání, které umožňovaly diferencovaný přístup k výuce.

Úlohy byly testovány buď všemi čtyřmi skupinami, nebo jen vybranými skupinami, což umožnilo širší spektrum ověření v praxi. Díky tomu si každý vyučující může flexibilně vybrat úlohy, které nejlépe odpovídají úrovni jeho žáků. V některých případech se ukázalo jako přínosné doplnit zadání o podrobnější instrukce, čímž se eliminovaly možné bariéry v pochopení úlohy. Významnou roli při snižování obtížnosti hrála také důkladná analýza zadání, která umožnila efektivnější postup při řešení úloh.

12 Závěr

Tato diplomová práce se zabývá problematikou rozvoje informatického myšlení žáků druhého stupně ZŠ prostřednictvím tvorby grafických výstupů v prostředí Pencil Code. Hlavním cílem bylo analyzovat informatické myšlení v tomto kontextu a na základě těchto poznatků navrhnout ucelený soubor aktivit, který byl následně pilotně ověřen v pedagogické praxi.

V teoretické části byla provedena analýza a komparace různých pojetí informatického a algoritmického myšlení se zaměřením na tvorbu grafických výstupů. Dále byly identifikovány možné nástroje a metody vhodné pro rozvoj informatického myšlení na úrovni základní školy. Tyto nástroje byly porovnány s prostředím Pencil Code.

V praktické části byl vytvořen soubor aktivit, který byl následně ověřen na vybraných skupinách žáků. Pilotní ověření ukázalo, že navržené úlohy efektivně podporují rozvoj informatického myšlení a algoritmických schopností žáků. Žáci si osvojili nejen práci se sekvencemi a cykly, ale také si procvičili dekompozici problémů, hledání vzorů a vizualizaci. Důležitou roli hrál také důkazný rozbor úlohy, který žákům pomohl efektivněji porozumět algoritmickému myšlení a snížit chybovost. Zároveň bylo potvrzeno, že možnost experimentování a kreativního rozvoje rovněž podporuje hlubší pochopení informatických principů.

Výsledky práce mohou sloužit jako podklad pro další rozvoj metodiky výuky algoritmizace a programování. Výstupy práce mohou být inspirací pro učitele při tvorbě vlastních vzdělávacích materiálů.

Soubor úloh je volně dostupný na adrese: https://gawkarol.pencilcode.net/home/index.html

13 Seznam použité literatury

BARR, V. a C. STEPHENSON. 2011. Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? [online]. [cit. 2025-04-10]. Dostupné z: https://www.researchgate.net/publication/247924673

BBC. 2024. Abstraction. [online]. 2024 [cit. 2024-02-07]. Dostupné z: https://www.bbc.co.uk/bitesize/guides/zttrcdm/revision/1.

BBC. 2024. Decomposition. [online]. 2024 [cit. 2024-02-07]. Dostupné z: https://www.bbc.co.uk/bitesize/guides/zqqfyrd/revision/1.

BBC. 2024. Introduction to computational thinking. [online]. 2024 [cit. 2024-02-07]. Dostupné z: https://www.bbc.co.uk/bitesize/guides/zp92mp3/revision/1.

BBC. 2024. Pattern recognition [online]. 2024 [cit. 2024-02-07]. Dostupné z: https://www.bbc.co.uk/bitesize/guides/zxxbgk7/revision/1.

BERRY, M. (2014) Computational Thinking in Primary Schools. [online]. [cit. 02. 20. 2025] Dostupné z: http://milesberry.net/2014/03/computational-thinking-in-primary-schools/.

Brian Ramirez. Pencil Code [online]. 2024 [cit. 2024-02-24]. Dostupné z: https://edtechbooks.org/onlinetools/pencil-code.

Brooks, R. „What is computational thinking," University of Yorks, [Online]. [cit. 2024-02-04]. Dostupné z: https://online.york.ac.uk/what-is-computational-thinking/.

CAS (2015) Computational Thinking - a Guide for Teachers, strana 7. [online]. [cit. 02. 25. 2025] Dostupné z: https://community.computingatschool.org.uk/resources/2324/single.

ČERNOCHOVÁ, Miroslava, ŠTÍPEK, Jiří a VAŇKOVÁ, Petra. 2019. Programování ve Scratch II (projekty pro 2. stupeň ZŠ). [online]. [cit. 2025-04-13]. Dostupné z: https://imysleni.cz/images/vzdelavaci_materialy/scratch-2/_METODIKA_kompletn%C3%AD_-_Programov%C3%A1n%C3%AD_ve_Scratch_II_beta_verze.pdf

CHYTIL, M. Informace o informatice. VTM – Proč a nač je počítač, Praha: Mladá fronta, 1987/1.

DOHNAL, P. (2008). Programování a programovací jazyky ve vzdělávání na ZŠ [online]. Brno, 2009 [cit. 2024-02-04]. Dostupné z: http://is.muni.cz/th/72886/pedf_m/.

EDUSKOP. 2020. Abstrakce. [online] [cit. 2025-02-19]. Dostupné z: https://eduskop.cz/courses/course-v1:UWB+PRIM-02+2020/about.

EDUSKOP. 2020. Algoritmy. [online] [cit. 2025-02-21]. Dostupné z: https://eduskop.cz/courses/course-v1:UWB+PRIM-02+2020/about.

EDUSKOP. 2020. Dekompozice. [online] [cit. 2024-11-29]. Dostupné z: https://eduskop.cz/courses/course-v1:UWB+PRIM-02+2020/about.

EDUSKOP. 2020. Vzory a sekvence. [online] [cit. 2024-11-28]. Dostupné z: https://eduskop.cz/courses/course-v1:UWB+PRIM-02+2020/about.

ELLIOTT, J. Action-research: a framework for self-evaluation in schools. TIQL-Working Paper No. 1. Cambrige: Institute of Education, 1981.

FUTSCHEK, Gerald. (2006). Algorithmic Thinking: The Key for Understanding Computer Science. In. Lecture Notes in Computer Science 4226, Springer.

FUTSCHEK, Gerald. 2017. The Importance of Computational Thinking. [online]. [cit. 2025-04-13]. Dublin: CEPIS Council, Dostupné z: https://www.cepis.org/.../ComputationalThinkingFutschek1.pptx

GAWLOWSKÁ, Karolína. Možnosti využití aplikace Pencil Code ve výuce. [online]. [cit. 2025-04-13]. Praha: Univerzita Karlova, Pedagogická fakulta, 2022. Dostupné z: https://dspace.cuni.cz/handle/20.500.11956/176350

HAVELKOVÁ, H. (2008). Algoritmy [online]. Poslední aktualizace 2008-09-30, [cit. 2024-02-04]. Dostupné z: http://wvc.pf.jcu.cz/ki/data/files/93prg1.ppt.

INFORMATICKÉ MYŠLENÍ. 2018. Co je informatické myšlení?. [online]. [cit. 2025-03-30]. Dostupné z: https://www.imysleni.cz.

JANÍK, Tomáš. Akční výzkum pro učitele: Příručka pro teorii a praxi. [online]. [cit. 2025-03-24]. Brno: Pedagogická fakulta Masarykovy univerzity Katedra pedagogiky, 2003. Dostupné z: https://is.muni.cz/el/1441/jaro2006/ZS1BP_ZPM/um/um/TJ_akcni_vyzkum.pdf.

JANDOUREK, Jan. Sociologický slovník. Praha: Portál, 2001. ISBN 80-7178-535-0.

Jednotný metodický portál MŠMT. 2020. [online]. [cit. 2024-02-05]. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcovy-vzdelavacici programpro-zakladni-vzdelavani-rvp-zv/.

KEMP, P. 2014. Computing in the curriculum: Challenges and strategies from a teacher's perspective. [online]. [cit. 2025-04-10]. Dostupné z: https://eprints.soton.ac.uk/370403/1/Kemp.pdf

LODI, Michael a MARTINI, Simone. Computational Thinking, Between Papert and Wing. Online. Science & Education. 2021, č. 30, s. 883–908. [online]. [cit. 2025-03-30]. Dostupné z: https://doi.org/10.1007/s11191-021-00202-5.

MANĚNOVÁ, M. a PEKÁRKOVÁ, S. (2020). Rozvoj informatického myšlení s využitím robotických hraček v mateřské škole a na 1. stupni ZŠ. iMyšlení.cz. [online]. [cit. 20. 11. 2024] Dostupné z: https://imysleni.cz/ucebnice/rozvoj-informatickeho-mysleni.

MŠMT. (2018). Rámcový vzdělávací program pro obor vzdělání 18–20–M/01 Informační technologie [online]. [cit. 30. 1. 2025]. Dostupné z: https://www.edu.cz/rvpsov/ciste/18-20-M01.pdf.

MŠMT. (2007). Rámcový vzdělávací program pro gymnázia. [online]. [cit. 2025-01-30]. Dostupné z: https://www.edu.cz/wp-content/uploads/2020/08/RVPG-2007-07_final.pdf.

NEŠPOR, Zdeněk R. a BURIÁNEK, Jiří. Analýza obsahová. Online. Sociologická encyklopedie. 2017, 10. 11. 2018. [online]. [cit. 2025-03-24]. Dostupné z: https://encyklopedie.soc.cas.cz/w/Anal%C3%BDza_obsahov%C3%A1.

NEŠPOR, Zdeněk R.; LINHART, Jiří a VODÁKOVÁ, Alena. Metoda srovnávací. Online. Sociologická encyklopedie. 2017, 11. 12. 2017. [online]. [cit. 2025-03-24]. Dostupné z: https://encyklopedie.soc.cas.cz/w/Metoda_srovnávací.

RVP ZV. (2023). Rámcový vzdělávací program pro základní vzdělávání – edu.cz. [online]. [cit. 2025-01-30]. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcovy-vzdelavaci-program-pro-zakladni-vzdelavani-rvp-zv/.

RVP ZV. (2023). Změny v Rámcovém vzdělávacím programu pro základní vzdělávání. [online]. [cit. 2025-01-30]. Dostupné z: https://www.edu.cz/wp-content/uploads/2023/07/RVP_ZV_2023_zmeny.pdf.

SCHUBERT, S., Schwill, A. (2011). Didaktik der Informatik. 2. Aufl. Heidelberg: Spektrum Akademischer Verl, 2011. ISBN 9783827426529.

SEEBAUER, R. (Hg.). Akteulle Trends im europäischen Bildungswesen. Wien, Brno: Paido, 2002.

SKALKA, Ján et al. Algoritmizácia a úvod do programovania. Nitra: UNIVERZITA KONŠTANTÍNA FILOZOFA v NITRE, 2007, s. 158. ISBN: 9788080942175.

ŠTÍPEK, Jiří. 2020. Algoritmizace a programování. In: PEDF UK.

VANÍČEK, Jiří. 2018. Co je informatické myšlení? [online]. [cit. 03. 10. 2024] Dostupné z: https://imysleni.cz/informaticke-mysleni/co-je-informaticke-mysleni.

WING, J.M. 2006. Computational thinking. Carnegie Mellon University. [online]. [cit. 22. 11. 2024] Dostupné z: https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf.

WIRTH, Niklaus. Algoritmy a štruktúry údajov. 2. vyd. Bratislava: Alfa, 1989. ISBN 80-05-00153-3.

14 Vyjádření k využití nástrojů umělé inteligence

Prohlašuji, že při tvorbě této práce byly nástroje umělé inteligence využity pouze jako podpora pro stylistické úpravy, hlavně tedy pro gramatickou korekturu a návrhy formulací textu. Jejich použití plně odpovídalo platným předpisům a pravidlům, včetně Etického kodexu Univerzity Karlovy. Práce zároveň zachovává originalitu a integritu mé vlastní tvorby.

15 Seznam obrázků

16 Seznam zkratek

IM	Informatické myšlení
AM	Algoritmické myšlení
RVP	Rámcový vzdělávací program
ŠVP	Školní vzdělávací program
fd	příkaz jít rovně
bk	příkaz jít pozadu
rt	příkaz zatočit doprava
lt	příkaz zatočit doleva
rt,100	příkaz zatočit doprava o poloměr 100 bodů
lt,100	příkaz zatočit doleva o poloměr 100 bodů
fill	příkaz vybarvení obrazce
pen	příkaz pera
dot	příkaz vytvoření tečky
box	příkaz vytvoření kostičky
jumpto	skok na souřadnice x a y
moveto	přesun na souřadnice x a y
home	počáteční souřadnice a úhel obrazce želvy
speed	zrychlení pohybu postavy želvy
random	příkaz pro náhodnou proměnnou

17 Seznam tabulek